Популярные лекции
по математике

С. А. АБРАМОВ

ЭЛЕМЕНТЫ ПРОГРАММИРОВАНИЯ

АН
КНИГИ
НИКИТИНА

ПОПУЛЯРНЫЕ ЛЕКЦИИ ПО МАТЕМАТИКЕ
ВЫПУСК 56

С. А. АБРАМОВ

ЭЛЕМЕНТЫ ПРОГРАММИРОВАНИЯ

МОСКВА «НАУКА»
ГЛАВНАЯ РЕДАКЦИЯ
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ
1982

22.18
А 16
УДК 519.6

Абрамов С. А.

А16 Элементы программирования. — М.: Наука, 1982. — 96 с. — (Популярные лекции по математике). — 15 к.

Книга посвящена популярному изложению начальных сведений о программировании и программном обеспечении. Рассматриваются такие основные понятия, как алгоритм, алгоритмический язык, вычислительная машина, трансляция и операционная система.

Для чтения книги достаточно знаний в объеме программы средней школы.

$\text{А}\,\frac{1500000000-028}{053(02)-82}\,74\text{-}82$

ББК 22.18
519.6

$\text{А}\,\frac{1500000000-028}{053(02)-82}\,74\text{-}82$

© Издательство «Наука»; Главная редакция физико-математической литературы, 1982

ОГЛАВЛЕНИЕ

Предисловие 4
Введение 5

Глава I. Об алгоритмах решения задач 8

§ 1. Экономия операций 8
§ 2. Обозначения 10
§ 3. Повторения 13
§ 4. Условия 17

Глава II. Пример алгоритмического языка 20

§ 1. О записи программ. Выражения 20
§ 2. Операторы ввода, вывода и присваивания. Программа 23
§ 3. Условный и составной операторы 27
§ 4. Оператор перехода, пустой оператор 30
§ 5. Целый тип 33
§ 6. Массивы чисел 36
§ 7. Оператор цикла 39

Глава III. Вычислительная машина и ее входной язык . . . 43

§ 1. Память машины. Машинные слова 43
§ 2. Арифметические операции. Команды перехода 46
§ 3. Выполнение программы 49
§ 4. Регистры. Обработка массивов 52
§ 5. Операции над словами произвольного вида 56

Глава IV. Трансляция 61

§ 1. Преобразование последовательностей символов . . . 61
§ 2. Вычисление значения выражения 63
§ 3. Трансляция выражений 68
§ 4. Трансляция операторов и программы 71

Глава V. Диалоговые программы. Операционная система . . 75

§ 1. Диалоговые программы 75
§ 2. Обмен сообщениями в диалоге 77
§ 3. Многопрограммный режим работы машины 79

Дополнение. О доказательстве свойств программ . . 85

Литература для дальнейшего чтения 95

ПРЕДИСЛОВИЕ

Знакомство с кругом вопросов, затронутых в предлагаемой книжке, необходимо для того, чтобы составить представление о программировании.

Первые три главы посвящены основным понятиям программирования — понятиям алгоритма, программы, алгоритмического языка, входного языка вычислительной машины. В гл. IV рассматриваются методы перевода с алгоритмического языка на входной язык вычислительной машины. Эта глава труднее других, и при первом чтении можно ограничиться ее просмотром. В гл. V речь идет о диалоговых программах (игровых программах, программах продажи авиационных билетов и т. д.) и о многопрограммном режиме работы вычислительной машины (строении операционной системы).

В Дополнении рассказывается о том, каким образом могут доказываться различные свойства программ.

Сжатость изложения позволит читателям сберечь время, но потребует взамен некоторого напряжения внимания.

Книжка рассчитана на читателей, интересующихся началами программирования. Кроме представления о системе счисления по целому основанию $q \geqslant 2$ (о таких системах в серии «Популярные лекции по математике» имеется книга С. В. Фомина «Системы счисления». — М.: Наука, 1974), от читателей не требуется никаких знаний, выходящих за пределы школьного курса математики.

ВВЕДЕНИЕ

Прежде чем подойти к рассматриваемым в программировании задачам, вспомним некоторые задачи из школьной математики. В геометрических задачах на построение требуется предложить такой способ построения определенных фигур, который основывается исключительно на оговоренных заранее операциях — как правило, на тех операциях, которые можно выполнить с помощью циркуля и линейки. Способ построения должен охватывать возможные случаи взаимного расположения фигур, образующих исходные данные. Такие задачи известны и в алгебре — например, получение корней квадратного уравнения, заданного своими коэффициентами, при помощи операций сложения, вычитания, умножения, деления и извлечения квадратного корня. Вывод соответствующих формул дает решение задачи в общем виде.

В алгебре обычно не пользуются термином «построение», а говорят о вычислении, но тем не менее между подобными геометрическими и алгебраическими задачами есть существенное сходство: в тех и других рассматривается некоторое множество — геометрических фигур или чисел — и оговариваются операции, с помощью которых можно из имеющихся элементов получать новые; требуется указать способ построения удовлетворяющего некоторому условию элемента множества (или группы элементов). Задачи на построение можно рассматривать применительно к самым разным множествам с наборами операций.

Четко описанный способ построения (в частности, вычисления) называется *алгоритмом*, этот термин получил широкое распространение в последние десятилетия.

Поиски алгоритмов решения различных задач увлекали математиков во все времена. Уже в древности были открыты способы построения середины отрезка и

биссектрисы угла с помощью циркуля и линейки, получены формулы площадей и объемов геометрических фигур и тел, правила выполнения арифметических действий над числами в десятичной системе. Решения многих подобных задач имеют большое практическое значение.

В настоящее время интерес к алгоритмам особенно велик из-за возможности использования в научных исследованиях, технике, экономике и т. д. вычислительных машин — специальных автоматов, выполняющих построение некоторых величин в точном соответствии с указанным алгоритмом. Эта возможность привлекательна по той причине, что явления и процессы, которые изучаются в рамках упомянутых исследований, часто удается описать с помощью понятий математики — множеств, функций, систем уравнений, неравенств и т. д. — и для получения конкретных сведений об изучаемых явлениях и процессах надо произвести некоторые действия над математическими объектами. Человеку достаточно описать алгоритм необходимых преобразований, вычислений и т. д., а сами эти действия (как правило, настолько обширные и громоздкие, что их невозможно выполнить вручную) выполнит машина.

Построения с помощью циркуля и линейки давно перестали представлять практический интерес, и, конечно для их выполнения вычислительные машины не нужны. Но аналогия процесса составления алгоритма с решением геометрической задачи на построение полезна для уяснения как характера этого процесса, так и самого понятия алгоритма.

Не следует думать, что вычислительные машины выполняют только численные алгоритмы: эти машины могут преобразовывать и получать последовательности символов — например, алгебраические формулы, тексты и т. д. (числа, заданные наборами цифр, — частный случай последовательностей символов).

Правила записи алгоритма для выполнения его вычислительной машиной оказываются очень жесткими — автомат не может ничего додумывать за человека. Совокупность средств и правил записи алгоритмов называется *алгоритмическим языком.*

Алгоритм, записанный на понятном вычислительной машине алгоритмическом языке, называется *программой.*

Соответственно говорят о *программировании*, заключающемся в создании и исследовании программ для вычислительных машин. Для удобства программирования предлагаются новые алгоритмические языки; это ставит задачу разработки алгоритмов перевода программ на этих языках в такие программы, которые понятны вычислительной машине в силу ее конструкции. Алгоритм перевода записывается в виде программы, после чего перевод может выполнить сама вычислительная машина. Имеется много программ, которые, подобно этим программам-переводчикам, предоставляют различные удобства программистам, расширяют возможности вычислительной машины. Такие программы образуют *программное обеспечение* вычислительной машины.

Благодаря соответствующему программному обеспечению становится возможным составление программ, управляющих диалогом между человеком и вычислительной машиной. Сюда относятся различные игровые программы, программы, автоматизирующие справочные службы (машина дает ответ на вопрос человека) и продажу авиационных и железнодорожных билетов.

ГЛАВА I

ОБ АЛГОРИТМАХ РЕШЕНИЯ ЗАДАЧ

В настоящей главе рассматривается важнейшее понятие программирования — алгоритм, т. е. способ построения одних величин, исходя из других, с помощью данных операций. При решении задач всегда имеется в виду, что практический интерес представляют экономные алгоритмы.

§ 1. Экономия операций

Если предстоит выполнить работу, сводящуюся к последовательности некоторых знакомых операций, и если мы хотим сделать все необходимое с меньшей затратой сил и времени, то надо продумать такой порядок выполнения работы, который позволит обойтись возможно меньшим количеством операций. Разберем математический пример.

Пусть дано некоторое число a и натуральное число n. Способ вычисления a^n, основанный на формуле

$$a^n = \underbrace{a \cdot a \cdot \ldots \cdot a}_{n},$$

годится в любом случае, но слишком разорителен. Например, a^4 экономнее вычислять как $(a^2)^2$. Как для данного n решить задачу, выполнив наименьшее число умножений?

Если накоплен некоторый запас значений степеней величины a, например $a^{m_1}, a^{m_2}, \ldots, a^{m_p}$, то, выполнив одно умножение, можно пополнить его любым из значений $a^{m_i+m_j}$, $1 \leqslant i \leqslant j \leqslant p$. Первоначально запас состоит только из a^1, и он пополняется одним умножением до a^1, a^2. Это изобразим графически с помощью рис. 1.

От a^1, a^2 одним умножением можно перейти либо к a^1, a^2, a^3, либо к a^1, a^2, a^4; графическое изображение

см. на рис. 2, каждый путь от вершины 1 к вершине m задает способ вычисления a^m.

Так можно наращивать это «дерево» и дальше: если $i_1, i_2, \ldots, i_k$ — путь от вершины $i_1 = 1$ к концевой вершине i_k, то от вершины i_k отводим отрезки к новым вершинам $i_r + i_s$, причем берем только не равные между

Рис. 1.

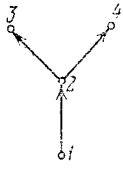

Рис. 2.

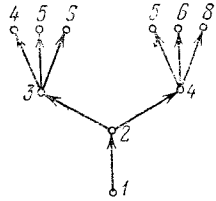

Рис. 3.

собой значения, большие i_k. «Дерево», изображенное на рис. 2, без труда превращается в «дерево» на рис. 3.

Итак, двумя умножениями могут быть вычислены a^3, a^4, тремя умножениями — a^4, a^5, a^6, a^8 (a^5 и a^6 — двумя способами). Чтобы найти способ скорейшего вычисления a^n, надо наращивать «дерево» ярус за ярусом до появления вершины n. Наращивая «дерево», изображенное на рис. 3, еще на ярус, получаем, например, число

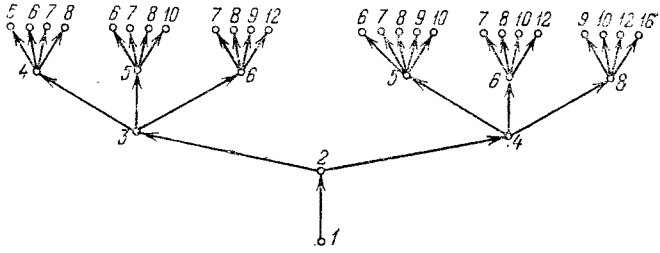

Рис. 4.

$n = 10$ (рис. 4). Получается, что a^{10} невозможно вычислить тремя умножениями, но четырех умножений достаточно. Для этого существуют четыре способа:

$$a^1,\ a^2,\ a^3,\ a^5,\ a^{10};$$
$$a^1,\ a^2,\ a^4,\ a^5,\ a^{10};$$
$$a^1,\ a^2,\ a^4,\ a^6,\ a^{10};$$
$$a^1,\ a^2,\ a^4,\ a^8,\ a^{10}.$$

Нам удалось перебрать все способы экономного вычисления, которое можно успешно использовать в вычислительной практике. Здесь и далее мы разделяем разработку способа вычислений, с одной стороны, и фактическое выполнение вычислений, с другой. Предметом наших занятий будет исключительно разработка способов вычислений — достижения современной техники позволяют переложить выполнение самих вычислений на специальные вычислительные машины. Последнее не снижает важности усилий человека по экономии операций. Как бы быстро ни работала вычислительная машина, время ее работы в некоторых случаях все-таки может оказаться слишком большим, чтобы считать его приемлемым: это могут быть месяцы или годы.

Опасность больших затрат времени возникает при решении сложных задач, но экономное решение элементарной задачи тоже может сыграть существенную роль, если она многократно повторяется при решении сложной задачи.

Задачи.

1. Показать, что при неограниченном наращивании «дерева» любое число n появляется лишь конечное число раз.

2. Сколько существует способов вычисления a^5 (путей от 1 к 5)?

3. Каково наименьшее число умножений, необходимых для вычисления a^{15}?

Рис. 5.

4. Пусть $n = 2^m$. Каково наименьшее число умножений, необходимое для вычисления a^n?

5. Из ящика, в котором лежат 5 белых и 3 черных шара, один за другим извлекаются 6 шаров. Построить «дерево» возможных извлечений. Начинается построение с «дерева», соответствующего извлечению первого шара (рис. 5).

§ 2. Обозначения

Рассмотрим следующий порядок вычисления a^{10}: a^1, a^2, a^3, a^5, a^{10}. Хотелось бы описать его точнее, показывая, из каких сомножителей получается каждая величина. Обычный прием для этого — введение буквенных обозначений всех промежуточных величин и опи-

сание вычисления очередной величины с помощью уже введенных обозначений.

Чтобы ввести обозначение, например u, для значения выражения $v \cdot w$, будем писать

$$u := v \cdot w.$$

Последнюю запись следует понимать как *инструкцию* (приказ) следующего содержания: вычислить значение $v \cdot w$ и обозначить его u.

Вычисление a^{10} описывается четырьмя инструкциями:

$$b := a \cdot a; \quad c := b \cdot a; \quad d := b \cdot c; \quad e := d \cdot d.$$

Ясно, что $b = a^2$, $c = a^3$, $d = a^5$, $e = a^{10}$. После вычисления d, т. е. a^5, величины $b = a^2$ и $c = a^3$, сыграв свою роль, становятся ненужными. Поэтому все вычисление можно описать, например, так:

$$b := a \cdot a; \; c := b \cdot a; \; c := b \cdot c; \; c := c \cdot c.$$

Выполнение третьей инструкции ведет:

1) к вычислению $b \cdot c$ (т. е. a^5);
2) к забвению прежнего значения c (т. е. a^3);
3) к обозначению через c вновь вычисленного значения.

Аналогично, выполнение последней инструкции приводит к тому, что a^{10} обозначается через c.

Мы получаем возможность экономить обозначения. Порядок вычисления a^1, a^2, a^4, a^5, a^{10} предпочтительнее, чем a^1, a^2, a^3, a^5, a^{10}, — он описывается с единственным дополнительным обозначением:

$$b := a \cdot a; \quad b := b \cdot b; \quad b := b \cdot a; \quad b := b \cdot b.$$

Кроме a, на каждом этапе вычислений надо помнить только одно число, обозначенное через b.

Хотя память (специальное электронное устройство) современных вычислительных машин способна вместить большое число величин, решение сложных задач часто упирается именно в нехватку памяти. Поэтому экономия памяти в практическом отношении не менее важна, чем экономия операций.

Дальше будут разбираться задачи, подобные задаче вычисления a^{10}, в следующем смысле: в рассматриваемом множестве можно выполнять над элементами некоторые операции; требуется описать способ построения оговоренного условием задачи элемента. Именно, в задаче вычисления a^{10} рассматривалось множество чисел с операцией умножения. Требовалось описать способ вы-

числения десятой степени данного элемента рассматриваемого множества, т. е. составить алгоритм.

Задачи на построение хорошо известны из геометрии. Там рассматривается множество элементарных геометрических объектов на плоскости (точек, прямых, окружностей), в котором, по предположению, можно выполнять известные операции: выделение общих точек, проведение прямой через две точки, проведение окружности с центром в выделенной точке радиусом, равным расстоянию между двумя выделенными точками, и т. д. Пример задачи вычисления a^n показывает, что геометрия — не единственная область математики, способная предлагать задачи на построение. Программирование как раз и занимается задачами на построение *). Решить задачу по программированию — это значит составить алгоритм.

Думая о возможности практического применения алгоритма, программист старается сделать алгоритм более экономным и по трудоемкости (числу операций), и по расходу памяти. Если алгоритм будет выполняться вычислительной машиной, то он должен быть записан так, чтобы машина была в состоянии, прежде всего, его прочесть и понять.

Позднее о вычислительных машинах и о средствах записи алгоритмов будет рассказано достаточно подробно. Пока же надо выяснить, на какого рода инструкции распадается алгоритм. То, что инструкций вида $u := v * w$, где $*$ — знак некоторой операции, недостаточно, будет ясно уже из следующего параграфа.

Задачи.

1. Показать, что для описания последовательного вычисления a^1, a^2, a^3, a^5, a^{10} недостаточно одного дополнительного обозначения.

2. Указать на «дереве», изображенном на рис. 4, путь, соответствующий следующему алгоритму вычисления a^{10}:

$$b := a \cdot a; \quad c := b \cdot b; \quad c := c \cdot c; \quad c := c \cdot b.$$

3. Описать алгоритм экономного вычисления a^{15}.

4. Рассмотрим множество целых чисел с операциями сложения и умножения. Пользуясь инструкциями вида $u := v * w$, где $*$ — это знак сложения $+$ или знак ум-

*) Множество вместе с фиксированным набором операций в программировании принято сейчас называть *типом*.

ножения ·, записать алгоритм вычисления значения y по формуле $ax^3 + bx^2 + cx + d$ так, чтобы не потребовалось более шести операций и более одного дополнительного обозначения.

5. Рассмотрим множество T всех подмножеств некоторого множества Ω вместе с известными операциями объединения, пересечения и разности. Пусть даны A, B, $C \in T$. Описать алгоритм построения $D \in T$, содержащего те элементы Ω, каждый из которых принадлежит ровно одному из A, B, C. То же самое — ровно двум.

§ 3. Повторения

Предложенные алгоритмы вычисления a^{10} универсальны в том смысле, что a можно придавать разные значения. Предварительное тщательное изучение соответствующего «дерева» окупается тем, что получен экономный алгоритм, который можно многократно использовать для разных a. Что представляет собой алгоритм вычисления a^n с произвольным натуральным показателем n?

Если пойти прежним путем минимизации умножений, то алгоритм должен включить в себя описание процесса наращивания «дерева», это обещает много хлопот (а если считать, что все построения ведутся только в множестве целых чисел, то надо еще придумать, как записывать «дерево» в виде набора чисел). «Дерево» быстро разрастается, и при выполнении такого алгоритма на манипуляции с «деревом» будет потрачено больше времени и памяти, чем при простейшем вычислении $\underbrace{a \cdot a \cdot \ldots \cdot a}_{n}$.

Займемся пока описанием простейшего алгоритма; остроумный алгоритм, который требует существенно меньше $n - 1$ умножений и не предполагает никаких действий с «деревом», рассмотрим в следующем параграфе.

В математике алгоритмы, подобные алгоритму вычисления a^n, записываются с помощью многоточия или каких-либо слов вроде «и так далее», потому что число операций, которые надо выполнять, существенно зависит от данных задачи, в нашем случае от n.

Чтобы описание алгоритмов было более точным, мы воспользуемся циклическими инструкциями. Так, для

того чтобы заставить некоторую инструкцию P выполняться n раз подряд, будем писать

n раз повторять P

и эту запись назовем *циклической инструкцией.* С привлечением такой инструкции алгоритм вычисления изображается так:

$b := 1$; *n раз повторять* $b := b \cdot a$

При выполнении выписанного алгоритма с обозначением b будут последовательно связываться значения 1, a, a^2, ..., a^n. Если нужно, чтобы n раз повторилось выполнение последовательности инструкций P_1; P_2; ...; P_m, мы будем писать

n раз повторять $(P_1; P_2; \ldots; P_m)$

Например, алгоритм вычисления $n!$ можно изобразить так:

$p := 1$; $i := 0$; *n раз повторять*
$(i := i + 1;\ p := p \cdot i)$

При выполнении последней циклической инструкции i будем поочередно обозначать 1, 2, ..., n, а p — поочередно обозначать 1, $1 \cdot 2$, ..., $1 \cdot 2 \cdot \ldots \cdot n$. Без инструкций $p := 1$; $i := 0$, предшествующих в алгоритме циклической инструкции, обойтись нельзя: если p и i ничего не обозначают, то невозможно выполнить инструкции $i := i + 1$; $p := p \cdot i$.

Пусть разрешается пользоваться операциями умножения, деления и сложения. Алгоритм вычисления суммы $\frac{x}{1!} + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!}$, на первый взгляд, должен предписывать поочередное получение слагаемых возведением x в соответствующую степень, вычислением факториала и делением одного значения на другое. Но желание составлять по возможности экономные алгоритмы обязывает нас быть менее расточительными. Надо стараться так применять циклическую инструкцию, чтобы каждый следующий шаг максимально использовал сделанное на предыдущих шагах. Если уже получено $\frac{x^{i-1}}{(i-1)!}$, то для вычисления $\frac{x^i}{i!}$ достаточно домножить предыдущий результат на $\frac{x}{i}$.

Естественным будет следующее решение:

$i := 1,\ y := x;\ s := x;\ n - 1$ *раз повторять*

$$\left(i := i + 1;\ y := y \cdot \frac{x}{i};\ s := s + y\right).$$

Решая задачу, программист в большинстве случаев не стремится полностью минимизировать число операций и обозначений, а просто не позволяет себе заведомых излишеств. Анализ, подобный проведенному нами для задачи вычисления a^{10}, в более сложных случаях может оказаться невозможным. Кроме того, хитроумная экономия операций часто требует дополнительного расхода памяти.

Приходится иметь дело и с алгоритмами, при описании которых тоже надо бы употребить циклическую инструкцию, но число шагов в ней определяется непосредственным рассмотрением исходных данных.

Алгоритм Евклида нахождения наибольшего общего делителя натуральных чисел a и b $(a > b)$ НОД(a, b) основывается на соотношении

$$\text{НОД}(a, b) = \text{НОД}(b, r(a, b)),$$

где $r(a, b)$ — остаток от деления a на b, и на очевидном равенстве НОД$(a, 0) = a$. Например,

$$\text{НОД}(25, 15) = \text{НОД}(15, 10) = \text{НОД}(10, 5) = \text{НОД}(5, 0) = 5$$

Для описания подобных алгоритмов нужен другой вид циклической инструкции:

пока B *повторять* P

где B — некоторое условие (равенство или неравенство), а P — это или инструкция, или взятая в скобки последовательность инструкций; как только окажется, что B неверно, выполнение циклической инструкции прекратится.

Вот полное описание алгоритма Евклида:

пока $b \neq 0$ *повторять* $(c := r(a, b);\ a := b;\ b := c)$

В результате выполнения этой инструкции значения a и b будут изменены. Значение b станет равным нулю, значение же a станет равным наибольшему общему делителю первоначальных значений a и b. Если, например,

было $a = 25$, $b = 15$, то при выполнении выписанной циклической инструкции с обозначением c последовательно связываются значения 10, 5, 0, с обозначением a — значения 15, 10, 5, с обозначением b — значения 10, 5, 0. Описывая алгоритм Евклида, мы предполагали, что можно пользоваться операциями нахождения остатка и сравнения данного числа с нулем.

Задачи.

Считается, что можно пользоваться следующими операциями над числами: сложение, вычитание, умножение, деление, нахождение остатка и частного от деления с остатком, сравнение.

1. Описать алгоритм вычисления суммы

$$\frac{x}{1!} - \frac{x^2}{2!} + \frac{x^3}{3!} - \ldots \pm \frac{x^n}{n!};$$

знак последнего слагаемого определяется четностью n.

2. Описать алгоритм вычисления суммы

$$x^{(1^2)} + x^{(2^2)} + \ldots + x^{(n^2)}.$$

Указание. Значение x^{i^2} получается из $x^{(i-1)^2}$ домножением на x^{2i-1}. Значение x^{2i-1} получается из $x^{2(i-1)-1}$ домножением на x^2.

3. Описать алгоритм вычисления первого элемента последовательности

$$1, 1 + \frac{1}{2}, \quad 1 + \frac{1}{2} + \frac{1}{3}, \ldots,$$

большего данного числа a.

4. Описать алгоритм вычисления суммы цифр данного натурального числа n в системе счисления с основанием p.

5. Рассмотрим множество S всех конечных множеств натуральных чисел. Рассмотрим операцию, применение которой к $A \in S$ дает множество, состоящее из одного-единственного элемента, равного наибольшему элементу множества A (обозначение: $s(A)$). Кроме того, допустимы операции разности множеств и операция сравнения множества с пустым ($A = \varnothing$, $A \neq \varnothing$). Описать алгоритм, применимый к любому непустому $A \in S$, выполнение которого дает множество, состоящее из одного-единственного элемента, равного наименьшему элементу A.

§ 4. Условия

В этом параграфе при составлении алгоритмов принимается без оговорок, что допустимы арифметические операции сложения, вычитания, умножения, деления, нахождения частного и остатка (для целых неотрицательных операндов) и операции сравнения чисел между собой.

Идея одного достаточно экономного алгоритма вычисления a^n, не требующего рассмотрения «дерева», заключена в следующем: если $n = 2k$, то $a^n = (a^2)^k$; если же $n = 2k + 1$, то $a^n = (a^2)^k \cdot a$. Степень с показателем k вычисляется с учетом этих же соображений. Итак, надо разделить n на 2 с остатком $n = 2k + l$ $(0 \leqslant l \leqslant 1)$, потом это же проделать с k и т. д. Эти действия приводят, как известно, к построению двоичной записи n.

Так вот, предлагается следующий алгоритм вычисления a^n. Если $\alpha_m\alpha_{m-1} \ldots \alpha_1\alpha_0$ — двоичная запись n, то последовательно вычисляются значения a^{α_0}, $a^{\alpha_1\alpha_0}$, ..., $a^{\alpha_m\alpha_{m-1}\cdots\alpha_1\alpha_0}$. Для этого вычисляется цифра за цифрой двоичная запись n и, параллельно, значение за значением $a^{(2^0)}$, $a^{(2^1)}$, $a^{(2^2)}$, ... — каждое следующее значение получается возведением в квадрат предыдущего. Подсчитывается произведение тех значений, для которых соответствующая цифра двоичного представления равна 1.

Пример. Запись числа 9 в двоичной системе: 1001 $(9 = 1 \cdot 8 + 0 \cdot 4 + 0 \cdot 2 + 1)$; для вычисления a^9 достаточно найти a^2, a^4, a^8 (3 умножения), а затем определить $a^1 \cdot a^8$ (1 умножение).

Преимущество этого алгоритма в сравнении с простейшим: простейший алгоритм требует числа умножений, растущего как линейная функция от n, а здесь число умножений, грубо говоря, пропорционально количеству цифр числа n или двоичному логарифму n.

Словесное описание алгоритма содержит многоточия, которые мы исключаем, вставляя в алгоритм определенную в предыдущем параграфе циклическую инструкцию ($q(x, y)$ и $r(x, y)$ будут обозначать частное и остаток от деления x на y):

$c := 1$; $b := a$; *пока* $n > 0$ *повторять*

($l := r(n, 2)$; $n := q(n, 2)$; *если* $l = 1$, *то*

$c := c \cdot b$; $b := b \cdot b$)

Здесь пришлось воспользоваться новым видом инструкции: *если ..., то ...* Благодаря этому домножение c на b происходит, только когда остаток от деления n на 2 получается равным 1.

Инструкция такого вида называется условной, в общем случае она выглядит так:

если B, *то* P

или

если B, *то* P *иначе* Q

где B — некоторое условие, а P и Q — либо инструкции, либо взятые в скобки последовательности инструкций. Например, алгоритм вычисления наименьшего из x, y записывается одной инструкцией:

если $x < y$, *то* $z := x$ *иначе* $z := y$

Когда к словам *то* или *иначе* относится несколько инструкций, эти инструкции должны заключаться в скобки.

Напишем алгоритм подсчета количества отрицательных элементов в наборе чисел $a_1, \ldots, a_n$, где $a_i = \sin i$ (считаем, что синус — допустимая операция). Искомая величина обозначается через m.

$i := 1$; $m := 0$; n *раз повторить*

(*если* $\sin i < 0$, *то* $m := m + 1$; $i := i + 1$)

Еще пример: алгоритм распознавания, является ли данное натуральное число удвоенным нечетным; k получает значение 1, если ответ положительный, и 0 — в противном случае:

если $r(n, 2) \neq 0$, *то* $k := 0$ *иначе если*

$r(n, 4) = 0$, *то* $k := 0$ *иначе* $k := 1$

Задачи.

1. Показать, что предложенный в этом параграфе алгоритм вычисления a^n для некоторых n требует числа умножений, превышающего минимальное.

Указание. Рассмотреть $n = 15$.

2. Описать алгоритм вычисления количества единиц в записи данного натурального n в p-ичной системе.

3. Описать алгоритм вычисления суммы положительных и числа отрицательных элементов среди чисел $\sin i$, $i = 1, 2, \ldots, n$.

4. Описать алгоритм распознавания, верно ли, что $a \leqslant b \leqslant c$ (a, b и c — данные числа).

5. Вновь рассмотрим множество T всех подмножеств некоторого множества Ω. Допустимы операции разности двух элементов T и сравнения элемента T с пустым множеством. Описать алгоритм, применимый к любым A, $B \in T$, в результате выполнения которого выясняется, верно ли, что $A \subset B$.

* * *

Читатель видел примеры задач по программированию, он получил представление и о средствах, которыми располагает программист, решая задачу (описывая алгоритм). Однако полученные представления приблизительны. Рассмотренные виды инструкций не образуют небходимого для работы минимума (хотя и могли бы составить его ядро). Как, например, описать алгоритм получения по данному n всех значений $i!$, $i = 1, 2, \ldots, n$? Кроме того, предложенные виды инструкций нуждаются в уточнении. Например, к чему приводит выполнение инструкций

$y := 3$; *если* $x \geqslant 0$, *то если* $x \leqslant 1$, *то* $y := 2$
иначе $y := 1$

Пусть $x = -1$, будет ли y равен 1 или 3? Все зависит от того, как понимать вторую инструкцию; к какому из двух «*если*» относится единственное «*иначе*»?

Если алгоритм предназначается для машины, то такие двусмысленности совершенно недопустимы. Необходимы достаточно богатые и однозначно понимаемые средства предетавления алгоритмов. Такие средства называют теперь алгоритмическими языками. Один из таких языков будет разобран в следующей главе.

ГЛАВА II

ПРИМЕР АЛГОРИТМИЧЕСКОГО ЯЗЫКА

Будет рассмотрен, с существенными сокращениями, язык алгол 60. Это международный алгоритмический язык, разработанный группой американских и европейских ученых в 1960 г. Многие современные вычислительные машины способны выполнять алгоритмы, записанные на этом языке.

Рассматриваемый сокращенный вариант языка алгол 60 в тексте будет именоваться *алголом*.

Всюду в этой главе слово «программа» означает программу на алголе.

§ 1. О записи программ. Выражения

Программа записывается в виде последовательности специальных символов. К их числу относятся буквы, цифры, знаки препинания, знаки операций и некоторые слова в английском написании (последние рассматриваются как самостоятельные неделимые символы). Подстрочные индексы, всякого рода надстрочные показатели не допускаются, никаких специальных знаков переноса не существует.

В качестве букв используются буквы латинского алфавита, в качестве цифр — арабские цифры.

Буквенному математическому обозначению в алголе соответствует *идентификатор* — любая конечная последовательность букв и цифр, обязательно начинающаяся с буквы. Примеры идентификаторов:

A, *b*, *x1*, *x2*, *pi*, *a1a2*.

Идентификатор, являющийся обозначением числа, называется переменной, а число, им обозначенное, — значением переменной. Числа записываются в десятич-

ной системе, вместо запятой пишется точка. Примеры:

0, 260, 3.1415, +0.567, —0.18

Переменные и числа — простейшие частные случаи выражения. Более сложные выражения строятся из чисел и переменных с помощью знаков операций сложения, вычитания, умножения, деления и возведения в степень. Эти знаки суть +, —, ×, /, ↑. Кроме того, в выражении могут быть использованы круглые скобки и некоторые функции.

Указанные знаки операций деления и возведения в степень /, ↑ позволяют записывать в одну строчку выражения, которые традиционно записываются с выходом из строки: y^3 и $\frac{a}{b}$ записываются как $y \uparrow 3$, a/b. Знак операции умножения в алголе нельзя опускать или изображать точкой. Допустимая для математического текста запись $0{,}5(x-1)(x-2)(x-3)$ в алголе должна выглядеть так: $0.5 \times (x-1) \times (x-2) \times (x-3)$.

Значение выражения может быть вычислено, исходя из значений входящих в него переменных. При вычислении действуют обычные правила старшинства операций: самая старшая — возведение в степень, следующие по старшинству — умножение и деление, затем — сложение и вычитание. Из двух операций одинакового старшинства первой выполняется та, знак которой в выражении встречается раньше. Круглые скобки служат, как обычно, для изменения естественного порядка выполнения операций: $\frac{x-y}{2}$ следует записать в виде $(x-y)/2$, а не $x-y/2$.

Квадратные, фигурные скобки для регулирования порядка операций применять нельзя.

В выражениях могут встретиться функции sin, cos, arctan, ln, exp, abs, entier, sqrt. Вслед за обозначением функции в круглых скобках указывается аргумент:

sin (*E*) — синус *E* (*E* радиан),

cos (*E*) — косинус *E* (*E* радиан),

arctan (*E*) — главное значение арктангенса *E*,

ln (*E*) — натуральный логарифм *E*,

exp (*E*) — показательная функция *E*,

abs (*E*) — абсолютная величина *E* (модуль *E*),

entier (E) — целая часть E, т. е. наибольшее целое, не превосходящее E,

sqrt (E) — квадратный корень из E.

Традиционная математическая запись этих функций:

$$\sin E,\ \cos E,\ \operatorname{arctg} E,\ \ln E,\ e^E,\ |E|,\ [E],\ \sqrt{E}.$$

Перепишем по правилам алгола выражение

$$0{,}3\left[\left(\frac{\sin^2 x-\cos^2 x}{\sin\frac{x+y}{2}}-e^{|\cos x+\sin x|}\right)\ln x-\sqrt{2x-y}\right],$$

получим

```
0,3 × ((( sin (x) ↑ 2 — cos (x) ↑ 2)/sin ((x + y)/2) —
      exp (abs (cos (x) + sin (x)))) × ln (x) — sqrt (2 × x — y)).
```

Записывая выражение, нельзя размещать рядом два знака арифметических операций; записи *a* ↑ — *2*, *3* × — *2*, *x1*/ — *x2* не являются выражениями, выражениями будут

a ↑ (— *2*), *3* × (— *2*), *x1* (— *x2*).

Задачи.

1. Какие из следующих записей являются идентификаторами?

x^3, *x3*, *x* · *3*, x_3, *100t*, *t100*, *max*

2. Записать по правилам алгола выражение

$$\frac{1{,}7\cdot\left[(\sin^2 2x+\cos^2 3x)\cdot\left(\left|a_1^2-a_2^3\right|+a^{2a}\right)-3a_1\right]}{xa_1\ a_2}+1{,}2.$$

3. Записать выражение по традиционным правилам:

```
2.1 × abs (x/2 + 3 ↑ x ↑ 2) × (3.27 — (sin (x + 1) ↑ 3 +
                  abs (y) ↑ 3)) / (m × n) × k ↑ 2 × x ↑ (2 × x).
```

4. Вычислить значение выражения

```
(sin (x ↑ 2 — 1) ↑ 2) + cos (x ↑ 3 — 1) ↑ 3 ↑ abs (x — 2))/y ↑ 3 ×
                                        (3 ↑ 2 ↑ 2 + abs (y) ↑ 3)
```

при следующих значениях переменных: $x = 1$, $y = -2$.

5. В выражении

$$a/b \times c/d \times e/f \times h$$

расставить скобки так, чтобы выражению со скобками соответствовало

$$\frac{a}{b \cdot \dfrac{c}{d \cdot \dfrac{e}{f \cdot h}}}.$$

§ 2. Операторы ввода, вывода и присваивания. Программа

Запись алгоритма подразделяется на отдельные инструкции, которые в алголе принято называть *операторами.*

Для ввода данных и вывода результатов используются операторы ввода и вывода. Вот как, например, они могут выглядеть:

input (*x*)
input (*x1*, *x2*, *e*)
output (*x*)
output (*x*, *y*, *z*, *p*, *q*, *r*)

Оператор ввода состоит из идентификатора *input* (input по-английски означает ввод) и следующего за ним в круглых скобках списка переменных. Число переменных в списке не ограничено; если их больше одной, то они разделяются запятыми. Оператор вывода состоит из идентификатора *output* (output означает вывод) и следующего за ним в круглых скобках списка переменных.

При выполнении оператора ввода переменным присваиваются значения исходных данных. Программу, содержащую такие операторы, необходимо снабдить приложением — конечной последовательностью чисел, которую удобно рассматривать как очередь чисел (первый элемент — первый в очереди, второй элемент — второй в очереди и т. д.). Когда выполняется оператор ввода, переменным присваиваются очередные значения. Пусть приложение (очередь чисел) выглядит так:

3, 8.2, —1, 0.5,

тогда после выполнения оператора $input(a)$ переменной a будет присвоено значение 3. Если же через некоторое время будут выполнены операторы $input(b, c)$; $input(t)$, то b получит значение 8.2, c — значение -1, t — значение 0.5.

С помощью оператора вывода осуществляется вывод значений переменных. Выведенная последовательность чисел является результатом выполнения программы.

В результате выполнения оператора присваивания переменной присваивается значение некоторого выражения. Примеры операторов присваивания:

$$x1 := (-b + sqrt(b \uparrow 2 - 4 \times a \times c))/(2 \times a)$$

$$x2 := (-b - sqrt(b \uparrow 2 - 4 \times a \times c))/(2 \times a)$$

$$x := x + 1$$

Последний оператор предписывает увеличение x на единицу. Если до выполнения оператора $x := x + 1$ переменная x имела значение 1, то после выполнения этого оператора значение x станет равным 2.

Кроме операторов программа содержит описания. Описание может выглядеть, например, так *):

real *x1*

или

real *y, z, t*

В описании сообщается о том, какие переменные используются в программе.

Программа схематически может быть изображена следующим образом:

begin A; P_1; P_2; ...; P_k **end** **)

Здесь A — описание, а P_1, P_2, ..., P_k — операторы.

Пока оператор — это или оператор ввода, или оператор вывода, или оператор присваивания, а описа-

---

*) Специальные символы-слова в печатном тексте набираются жирным шрифтом, а в рукописном тексте подчеркиваются. Слово real означает по-английски действительный; далее переводы слов будут даваться в подстрочных примечаниях без указания на то, что слова английские.

**) begin — начало, end — конец.

ния — это символ **real**, за которым через запятую перечислены переменные; позднее понятия оператора и описания будут расширены. Далее без специальных оговорок считается, что при расширении понятий оператора и описания автоматически расширяются все те понятия, которые определялись раньше с помощью понятий оператора и описания.

Пример. Программа вычисления корней квадратного уравнения $ax^2 + bx + c = 0$, заданного коэффициентами a, b и c (предполагается, что $a \neq 0$ и что корни действительные):

```
begin real a, b, c, x1, x2;
    input(a, b, c);
    x1 := (— b + sqrt(b ↑ 2 — 4 × a × c))/(2 × a);
    x2 := (— b — sqrt(b ↑ 2 — 4 × a × c))/(2 × a);
    output(x1, x2)
end
```

При выполнении этой программы придется дважды вычислить значение $sqrt(b \uparrow 2 - 4 \times a \times c)$ и дважды — значение $2 \times a$. Поэтому более разумным вариантом будет

```
begin real a, b, c, x1, x2, d, e;
    input(a, b, c);
    d := sqrt(b ↑ 2 — 4 × a × c);
    e := 2 × a;
    x1 := (— b + d)/e; x2 := (— b — d)/e;
    output(x1, x2)
end
```

Если это почему-либо удобно, можно одно длинное описание заменить совокупностью более коротких. Во всех случаях описания располагаются в начале программы и отделяются друг от друга точкой с запятой.

Итак, возможен следующий вид программы:

begin A_1; A_2; ...; A_n; P_1; P_2; ...; P_k **end**

где A_1, A_2, ..., A_n — описания, а P_1, P_2, ..., P_k — операторы.

Второй вариант рассмотренной программы может быть переписан следующим образом:

```
begin real a, b, c; real x1, x2; real d, e;
    input(a); input(b, c);
    d := sqrt(b ↑ 2 — 4 × a × c); e := 2 × a;
    x1 := (— b + d)/e; x2 := (— b — d)/e;
    output(x1); output(x2)
end
```

Пример. Программа вычисления площади треугольника по трем сторонам:

```
begin real a, b, c, p, S;
    input(a, b, c);
    p := (a + b + c)/2;
    S := sqrt(p × (p — a) × (p — b) × (p — c));
    output(S)
end
```

Видно, что площадь вычисляется по формуле Герона.

При выполнении программы вычисления могут проводиться приближенно: числа в алголе имеют конечное число знаков после десятичной точки, и округления неизбежно возникают, например, при вычислении значений *sqrt*(*2*), *sin*(*1*) и т. д.

Задачи.

1. Написать программу вычисления значения квадратного трехчлена $ax^2 + bx + c$.

2. Написать программу нахождения среднего арифметического и среднего геометрического двух чисел.

3. Написать программу нахождения гипотенузы и площади прямоугольного треугольника по двум данным катетам.

4. Смешано v_1 литров воды температуры t_1 с v_2 литрами воды температуры t_2. Написать программу вычисления объема и температуры образовавшейся смеси.

5. Показать, что выполнение программы

```
begin real a, b, c, x;
    input(a, b, c);
    c := sqrt(b ↑ 2 — 4 × a × c);
    a := 2 × a;
    x := (— b + c)/a;
    b := (— b — c)/a;
    output(x, b)
end
```

приведет к тем же результатам, что и выполнение программ для решения уравнения $ax^2 + bx + c = 0$, разобранных в тексте параграфа. В новой программе вводится лишь одна дополнительная переменная, и при выполнении этой программы не повторяются уже проделанные вычисления.

§ 3. Условный и составной операторы

Если хотим присвоить переменной *max* наибольшее из значений переменных *x1* и *x2*, мы должны сравнить значения *x1* и *x2* и в зависимости от результата сравнения выполнить либо оператор $max := x1$, либо $max := x2$. Действия такого рода задаются *условным оператором*:

if B **then** P_1 **else** P_2 *)

где B — условие, P_1 и P_2 — операторы. Если условие удовлетворяется, то выполняется P_1, иначе выполняется P_2.

В качестве условий используются отношения. Отношения представляют собой записи равенств и неравенств. Примеры отношений:

$$a = b,\ d \neq 0,\ b \uparrow 2 - 4 \times a \times c > 0$$

В общем случае отношение — это два выражения, разделенных одним из знаков $=$, $\neq$, $\geqslant$, $>$, $<$, $\leqslant$. Для решения задачи о присваивании переменной *max* наибольшего из значений *x1* и *x2* достаточно выполнить условный оператор

if $x1 > x2$ **then** $max := x1$ **else** $max := x2$

Оператор, расположенный после **else**, может быть любым оператором; оператор, расположенный между **then** и **else**, не может быть условным.

Рассмотрим примеры программ, включающих в себя условные операторы. Пусть задана функция

$$y = \begin{cases} 0, & \text{если } x \leqslant 0; \\ x^2, & \text{если } x > 0. \end{cases}$$

*) if — если, then — то, else — иначе.

Требуется написать программу вычисления значения y по значению x. Программа может выглядеть так:

```
begin real x, y;
    input(x);
    if x ⩽ 0 then y := 0 else y := x ↑ 2;
    output(y)
end
```

Значение y зависит от значения x, и график зависимости приведен на рис. 6. Программа вычисления значения y по значению x:

```
begin real x, y;
    input(x);
    if x < 2 then y := x else
                 if x < 3 then y := 2 else y := x − 1;
                 output(y)
end
```

Если удовлетворено условие $x < 2$, то y получит значение, равное значению x, и это значение затем будет выведено. Если условие $x < 2$ не удовлетворяется, то значение y будет определено выполнением условного оператора

if $x < 3$ **then** $y := 2$ **else** $y := x - 1$

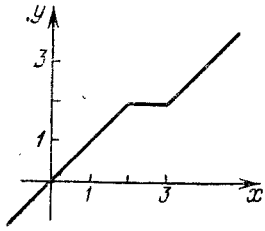

Рис. 6.

Допускается сокращенная форма условного оператора:

if B **then** P

где B — условие, P — оператор, не являющийся условным. В случае, если B удовлетворено, должен быть выполнен оператор P; если же B не удовлетворено, его выполнять не нужно.

Пример. Пусть даны два числа. Если первое больше второго по абсолютной величине, то необходимо уменьшить первое в пять раз. Иначе оставить числа без изменения. Программа:

```
begin real x, y;
    input(x, y);
    if abs(x) > abs(y) then x := x/5;
    output(x, y)
end
```

В условном операторе после **then** и **else** можно помещать по одному оператору. Однако часто необходимо в зависимости от результата проверки выполнить ту или иную группу операторов. Алгол предоставляет возможность сделать из группы операторов один *составной оператор*.

Структура составного оператора:

begin P_1; P_2; ...; P_k **end**

или

begin P **end**

где P, P_1, P_2, ..., P_k — любые операторы.

Существенно, что оператор **begin** P **end** при любом операторе P не является условным и может быть размещен после **then**.

Пусть квадратное уравнение $ax^2 + bx + c = 0$ задано коэффициентами a, b и c $(a \neq 0)$.

Ниже следует программа, при выполнении которой исследуется дискриминант уравнения, выводится число *0*, если дискриминант отрицателен, и выводится пара корней, если дискриминант неотрицателен.

```
begin real a, b, c, x1, x2, d;
    input(a, b, c);
    d := b ↑ 2 − 4 × a × c;
    if d < 0 then
    begin d := 0; output(d) end
          else
    begin d := sqrt(d); a := 2 × a;
    x1 := (− b + d)/a; x2 := (− b − d)/2;
    output(x1, x2) end
end
```

В этой программе в условном операторе после **then** и **else** размещены составные операторы.

Задачи.

1. Написать программу вычисления значения функции

$$y = \begin{cases} x^2, & \text{если } -2 \leqslant x \leqslant 2; \\ 4 & \text{в остальных случаях.} \end{cases}$$

2. Написать программу выбора наибольшего из трех чисел.

3. Показать, что если периодическая функция $f(x)$ с периодом t задана на отрезке $[0, t]$, то для любого x

значение выражения $x - entier(x/t) \times t$ принадлежит отрезку $[0, t]$ и значение функции для этого значения аргумента равно значению $f(x)$. Пользуясь этим, написать программу вычисления значения $f(x)$, период которой равен 4,5, а часть графика изображена на рис. 7.

4. Условный оператор

```
if x ↑ 2 > 2 then
    begin if x > 2 then y := x ↑ 3 else y := 8 end
          else y := 8 × x ↑ 2
```

устанавливает зависимость значения y от значения x. Построить график этой зависимости.

5. Написать программу полного исследования совокупности корней биквадратного уравнения $ax^4 + bx^2 + c = 0$. Должен быть выведен *0*, если действительных корней нет, и два или четыре корня в прочих случаях.

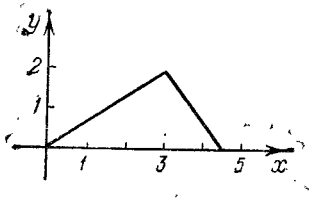

Рис. 7.

§ 4. Оператор перехода, пустой оператор

Каждый оператор может быть помечен *меткой* (идентификатором). Метка располагается перед оператором и отделяется от него двоеточием. Оператор *output*(x), помеченный меткой *finish*, запишется так:

finish: *output*(x)

Метка не влияет на выполнение оператора.

В рассмотренных программах операторы выполнялись в том порядке, в каком были написаны. Изменить этот порядок можно с помощью *оператора перехода*. Оператор перехода состоит из символа **go to** *), за которым следует метка. Примеры:

go to *finish*
go to *M*

Оператор перехода прерывает естественную последовательность выполнения операторов: следом за ним выполняется оператор, помеченный указанной меткой.

*) go to — перейти к.

Пусть программа содержит последовательность операторов:

$x := 2$; $A := B$; **go to** T; M: $A := 0$; $x := B$;

T: $y := x$; $x := 0$

В этом случае сначала выполняются операторы $x := 2$ и $A := B$. Затем следует переход к оператору, помеченному меткой T: $y := x$. После оператора $y := x$ будет выполнен оператор $x := 0$.

Напишем программу нахождения среди чисел

0.3, 0.3×1.3, $0.3 \times 1.3 \times 2.3$, ...

первого, большего данного числа a:

```
begin real a, b, c
    input (a);
    b := — 0.7; c := 1;
    repeat: b := b + 1; c := c × b;
    if a ⩾ c then go to repeat;
    output (c)
end
```

При выполнении этой программы переменная c последовательно принимает значения 0.3, 0.3×1.3, $0.3 \times \times 1.3 \times 2.3$, ... ; переменная b в свою очередь принимает значения 0.3, 1.3, 2.3, ... Изменение этих значений происходит до тех пор, пока значение c не станет больше значения a.

Перед оператором

repeat: $b := b + 1$

переменные b и c получают некоторые начальные значения, которые подобраны так, чтобы после первого выполнения операторов

repeat: $b := b + 1$; $c := c \times b$

переменные b и c приобрели значения, равные 0,3.

О пустом операторе. Пустой оператор ие предписывает никаких действий. По определению он представляет собой пустую совокупность символов. Как и все операторы, пустой оператор может быть помечен меткой.

В участке программы

$k := 0$; ;$m := 1$

между двумя операторами присваивания находится пустой оператор.

Рассмотрим конец некоторой программы или составного оператора

... *M*: **end**

Здесь перед символом **end** расположен помеченный меткой *M* пустой оператор. Основное назначение пустого оператора — дать возможность выхода из середины программы или составного оператора.

Последнюю программу из предыдущего параграфа можно переписать так:

```
begin real a, b, c, x1, x2, d;
    input (a, b, c);
    d := b ↑ 2 − 4 × a × c;
    if d < 0 then
    begin d := 0;
    output (d);
    go to finish
    end;
    d := sqrt (d);
    a := 2 × a;

    x1 := (− b + d)/a;
    x2 := (− b − d)/a;
    output (x1, x2);
    finish:
end
```

Задачи.

1. Выяснить, какие значения будут выведены в результате выполнения программы

```
begin real x, y;
    L: x := 0; y := x + 1;
    M: x := x + y; y := x + y; go to K;
    N: x := abs (x − 2); go to F;
    P: y := x; go to R;
    K: x := x ↑ 2; go to N;
    F: go to P;
    R: y := abs (x − y);
    S: output (x, y)
end
```

2. Написать программу нахождения среди чисел

$$1,\quad 1+\frac{1}{2},\quad 1+\frac{1}{2}+\frac{1}{3},\quad \ldots$$

первого, большего данного числа a.

3. Дано положительное число ε. Последовательность $a_1, a_2, a_3, \ldots$ образована по следующему закону:

$$a_i = \left(1-\frac{1}{2}\right)\left(1-\frac{1}{3}\right)\ldots\left(1-\frac{1}{i+1}\right).$$

Найти первый член a_n последовательности, для которого выполнено условие

$$|a_n - a_{n-1}| < \varepsilon.$$

Написать программу выполнения этого задания.

4. Может ли конец выполнимой программы иметь вид:

а) ; **go to** K **end**
б) **go to** K **end**
в) **go to** K; K: **end**
г) **go to** K; L: **end**

5. Воспользуясь приемом, продемонстрированным в последней программе этого параграфа, написать программу решения следующей задачи. Даны два числа; вывести первое число, если оно больше второго, и оба числа, если это не так.

§ 5. Целый тип

В § 2 было приведено схематическое изображение программы. Тогда нам было известно всего лишь три вида операторов. В дальнейшем понятие оператора существенно расширилось, теперь же у нас появится новый вид описания, в котором вместо **real** пишется **integer** *), например:

integer i, j

В данном случае мы имеем дело с описанием переменных целого типа (или переменных типа **integer**). Применявшиеся ранее описания были описаниями переменных действительного типа, или переменных типа **real**.

*) integer — целый.

Переменные типа **integer** могут принимать лишь целые значения, а переменные типа **real** — как целые, так и нецелые. Числа в алголе тоже разделяются на целые, или имеющие тип **integer**, и действительные, или имеющие тип **real**: если в записи числа использована точка, то оно действительное, в противном случае — целое.

Примеры целых чисел:

0, —3, 17, +0.193, —1000000, 5

Примеры действительных чисел:

3.14, —0.0001, 0.6, +17.0, —19.19191919, 5.0

Объясним причину, по которой выделяются величины типа **integer**. Уже говорилось, что значения арифметических выражений, в которые входят переменные и числа типа **real** и некоторые функции, могут вычисляться выполняющим программу приближенно, т. е. с небольшой погрешностью. Так,

0.1×10

может вычислиться как 0.999999999999, или 1.000000000001, или как что-нибудь в этом роде. Поэтому не имеет смысла проверять выполнение равенства между двумя действительными числами.

Арифметические операции сложения, вычитания, умножения и возведения в целую неотрицательную степень над целыми числами производятся абсолютно точно, и результатами этих операций снова являются целые числа. Значение функции *entier* всегда целое. Целочисленное значение выражения может быть сохранено в точном виде для дальнейших вычислений присваиванием этого значения переменной типа **integer**.

Программа вычисления $n!$:

```
begin integer n, i, p;
    input (n); p := 1; i := 0;
    L: i := i + 1; p := p × i;
    if i < n then go to L;
    output (p)
end
```

Здесь вычисление проводится за n шагов. Переменная принимает последовательно значения 1, 1×2, $1 \times 2 \times 3$, ..., $1 \times 2 \times 3 \times \ldots \times n$. Условный оператор обеспечивает контроль за тем, чтобы в произведение вошло ровно n сомножителей.

Если бы единственным описанием программы было **real** n, i, p, то, после того как в произведение вошло n сомножителей, переменная i могла бы иметь значение, чуть меньшее n, и в итоге имели бы приближенное значение $(n+1)!$.

Кроме названных ранее арифметических операций в алголе есть еще деление нацело. Для натуральных a и b это частное от деления a на b с остатком. В общем случае целых a, b, $b \neq 0$, определение операции таково:

$$a \div b = \begin{cases} \left[\frac{a}{b}\right], & \text{если } \frac{a}{b} \geqslant 0; \\ -\left[\left|\frac{a}{b}\right|\right], & \text{если } \frac{a}{b} < 0. \end{cases}$$

Эта операция одного старшинства с умножением и делением, что важно иметь в виду при вычислении значений выражений.

Операцию деления нацело можно использовать, чтобы узнать, кратно ли целое p целому q. А именно, p кратно q тогда и только тогда, когда $p \div q \times q = p$. Если p и q — натуральные, то остатком от деления p на q будет $p - p \div q \times q$.

Пример. Алгоритм Евклида нахождения НОД(a, b) для натуральных a и b.

```
begin integer a, b, c;
    input (a, b);
    L: c := a − a ÷ b × b; a := b; b := c;
    if b ≠ 0 then go to L;
    output (a)
end
```

Рассмотрим примеры программ, в которых встретятся величины разных типов.

Программа вычисления суммы $1 + \frac{1}{2} + \ldots + \frac{1}{n}$.

```
begin integer i, n; real s;
    input (n); s := 0; i := 0;
    L: i := i + 1; s := s + 1/i;
    if i < n then go to L;
    output (s)
end
```

Программа, в результате выполнения которой выясняется, имеются или не имеются среди чисел

$\cos(i^3)\sin\left(10000+\frac{i}{10000}\right)$, $i=0,\ 1,\ 2,\ \ldots,\ n$, меньшие $\frac{1}{10000}$. Если имеются, то выводится 1, если нет — 0.

```
begin integer i, n; input(n);
    i := -1;
    L: i := i + 1;
    if cos(i ↑ 3) × sin(10000 + i/10000) < 0.0001 then
    begin i := 1; go to finish end;
    if i < n then go to L;
    i := 0;
    finish: output(i)
end
```

Если оказывается, что некоторый член меньше $\frac{1}{10000}$, то следующие члены уже не рассматриваются.

Задачи.

1. Верно ли, что $b \times a \div b = a$ тогда и только тогда, когда a кратно b?

2. Дано натуральное n. Написать программу вычисления суммы всех чисел вида $i^3 - 3in^2 + n$ ($i = 1, 2, \ldots, \ldots n$), которые являются удвоенными нечетными числами.

3. Последовательность $u_0,\ u_1,\ u_2,\ \ldots$ определяется правилами

$$u_0 = 0,\quad u_1 = 1,\quad u_{i+2} = u_{i+1} + u_i \quad (i = 0,\ 1,\ 2,\ 3,\ \ldots).$$

Написать программу вычисления u_n.

4. Написать программу, в результате выполнения которой выясняется, есть ли среди чисел $i^3 - 17in^2 + n^3$ ($i = 1, 2, \ldots, n$) хотя бы одно число, которое кратно a и не кратно b.

5. Доказать, что любую целочисленную (в рублях) денежную сумму больше 7 руб. можно выплатить без сдачи трешками и пятерками. Написать программу выработки по данному целому числу $n > 7$ пары целых неотрицательных чисел a и b таких, что $n = 3a + 5b$.

§ 6. Массивы чисел

Переменные, снабженные индексом, широко используются в математике — например, координаты точки в пространстве часто обозначаются x_1, x_2, x_3. Алгол также предоставляет удобную возможность использования та-

ких переменных. Выражения $x[1]$, $x[2]$, ..., $x[n]$, где вместо x может стоять любой идентификатор, а вместо n — любое целое положительное число, называются *переменными с индексом.*

Выписанные переменные с индексом образуют *массив x*, идентификатор x называется *идентификатором массива.*

В операторе присваивания слева от знака $:=$ может быть и переменная с индексом: $b[1] := 55$; $b[2] := 125$.

Массивы можно вводить и выводить целиком с помощью операторов ввода и вывода: в соответствующем операторе указывается идентификатор массива.

Пусть a — простая переменная, b обозначает массив из 20 чисел, c — массив из двух чисел; тогда в результате выполнения операторов *input*(a, b) и *output*(a, c) будет введено одно число для переменной a, 20 чисел — для массива b и будет выведено значение переменной a и два числа — элементы массива c.

Описания массивов располагаются среди других описаний и содержат в себе сведения о том, сколько элементов в массиве и какого типа эти элементы. Примеры:

integer array $b[1:10]$
real array $c[1:2]$, $p[1:5]$
integer array t, x, $y[1:20]$*)

После **real array** или **integer array** пишется идентификатор массива, за которым в квадратных скобках указывается количество элементов в виде двух чисел, разделенных двоеточием, первое число — номер первого элемента, второе число — номер последнего; дальше через запятую пишутся другие идентификаторы массивов и указываются соответствующие количества элементов, если имеются еще массивы того же типа. Из приведенных выше примеров видно, что если несколько массивов одного типа имеют одно и то же количество элементов, то это количество можно указать один раз. Так, массивы t, x, y имеют по 20 элементов.

Описания

integer array a, $b[1:2]$, $c[1:5]$, p, $t[1:10]$
real array x, $y[1:20]$, $l[1:1]$, h, $g[1:18]$

---

*) array — массив.

дают следующие сведения. Целочисленные массивы a и b содержат по два элемента, c — пять элементов, p и t — по 10 элементов; действительные массивы x и y содержат по 20 элементов, l — один элемент $l[1]$, h и g — по 18 элементов. Переменная с индексом должна иметь индекс, величина которого не превышает номера последнего элемента массива, заданного в описании массива.

Индексом может быть не только целое число, но и произвольное выражение с целым значением. Например, переменные с индексами $b[i]$, $b[2\times i]$, $b[2\times i-1]$ удобно использовать для поочередного рассмотрения всех элементов массива, элементов, стоящих на четных и нечетных местах.

Здесь проявляется преимущество обозначений $b[1]$, $b[2]$, ... перед обозначениями $b1$, $b2$, ...

Программа нахождения наибольшего значения из a_1, a_2, ..., a_{90}:

```
begin integer i; real max; real array a[1 : 90];
    input (a);
    i := 1; max := a[1];
    L: i := i + 1; if a[i] > max then max := a[i];
    if i < 90 then go to L;
    output (max)
end
```

Значение наибольшего элемента определяется за 90 шагов. После выполнения i-го шага значение max равно наибольшему из a_1, a_2, ..., a_i.

Программа построения целочисленного массива a_1, a_2, ..., a_{200}, элементы которого соответственно равны значениям b_1, c_1, b_2, c_2, ..., b_{100}, c_{100}, где b_1, b_2, ..., b_{100} и c_1, c_2, ..., c_{100} — данные целочисленные массивы:

```
begin integer i, m; integer array c, b[1, 100],
                                    a[1 : 200];
    input (b, c);
    m := 0; i := 0;
    L: i := i + 1; m := m + 1; a[i] := b[m];
    i := i + 1; a[i] := c[m];
    if i < 200 then go to L;
    output (a)
end
```

Построение массива осуществляется за 200 шагов. На i-м шаге присваивается необходимое значение пере-

менной $a[i]$. Это присваивание производится по-разному в зависимости от четности i: если $i = 2m$, то $a_i = c_m$; если $i = 2m - 1$, то $a_i = b_m$.

Программа вычисления $a_1 b_{1000} + 2a_2 b_{999} + \ldots + 999a_{999} b_2 + 1000a_{1000} b_1$:

```
begin integer i; real s; real a, b[1 : 1000];
    input(a, b);
    i := 0; s := 0;
    L : i := i + 1; s := s + i×a[i]×b[1001 − i];
    if i < 1000 then go to L;
    output(s)
end
```

Задачи.

1. Написать программу вычисления суммы элементов данного массива $a_1, a_2, \ldots, a_{25}$.

2. Написать программу вычисления количества тех элементов целочисленного массива $n_1, n_2, \ldots, n_{100}$, которые являются удвоенными нечетными числами.

3. Дан массив $a_1, a_2, \ldots, a_{200}$. Написать программу построения массивов $b_1, b_2, \ldots, b_{100}$ и $c_1, c_2, \ldots, c_{100}$, элементы которых равны соответственно значениям $a_1, a_3, \ldots, a_{199}$ и $a_2, a_4, \ldots, a_{200}$.

4. Написать программу определения количества элементов в данном массиве $a_1, a_2, \ldots, a_{700}$, удовлетворяющих условию $0 < a_1 \leqslant i$.

5. Написать программу, после выполнения которой становится известным, имеются ли в целочисленном массиве $c_1, \ldots, c_{1000}$ два подряд идущих нулевых элемента.

§ 7. Оператор цикла

Для многократного повторения каких-либо действий часто используют *оператор цикла*

for $p := A$ **step** B **until** C **do** S*)

Буквой p обозначена некоторая переменная (без индекса), она называется параметром цикла, буквами A, B и C — выражения, буквой S — оператор. Предполагается, что A, B и C не зависят от p и что выполнение оператора S не приводит к изменению значений параметра цикла и выражений A, B, C.

*) for — для, step — шаг, until — до, do — выполнять.

Оператор цикла заставляет выполняться оператор S многократно. Сначала он выполняется для $p = A$, затем для $p = A + B$, затем для $p = A + 2B$ и т. д. для всех значений p вида $A + mB$, принадлежащих отрезку $[A, B]$ или $[B, A]$ в зависимости от того, $A \leqslant B$ или $B \leqslant A$.

Рассмотрим, например, оператор цикла

for $i := 1$ **step** 1 **until** 99 **do** $s := s + i \uparrow 2$

Если перед его выполнением переменная s имела значение 0, то после выполнения переменная s примет значение $1^2 + 2^2 + \ldots + 99^2$.

Программа вычисления $n!$:

```
begin integer i, n, p;
    input(n);
    p := 1;
    for i := 2 step 1 until n do p := p × i;
    output(p)
end
```

Программу можно написать и так:

```
begin integer i, n, p;
    input(n);
    p := 1;
    for i := n step - 1 until 2 do p := p × i;
    output(p)
end
```

При составлении программы вычисления суммы $1! + 2! + \ldots + n!$ естественно воспользоваться соотношением $i! = i(i-1)!$, $0! = 1$:

```
begin integer n, i, a, s;
    input(n); s := 0; a := 1;
    for i := 1 step 1 until n do
    begin a := a × i; s := s + a end;
    output(s)
end
```

Оператор S, выполняемый многократно, может содержать в себе оператор перехода. В этом случае выполнение оператора цикла может кончиться раньше, чем будет перебрана вся серия значений параметра цикла. Во всех случаях по окончании выполнения оператора

цикла значение параметра цикла считается неопределенным.

Программа нахождения индекса первого четного элемента среди элементов с нечетными индексами массива $a_1, a_2, \ldots, a_{1000}$. Если поиск окажется безуспешным, то надо вывести —1.

```
begin integer k, i; integer array a[1 : 1000];
    k := — 1;
    for i := 1 step 2 until 999 do
    if a[i] ÷ 2 × 2 = a[i] then
    begin k := i; go to L end;
    L: output (k)
end
```

Програма останется правильной, если ...**until** *999*... заменить на ... **until** *1000*...

Интересна конструкция, когда оператор в цикле сам является оператором цикла. Таких вложений операторов цикла может быть несколько. Рассмотрим следующий оператор:

```
for k := 1 step 1 intil 20 do
                for p := 1 step 1 until a do output (k, p)
```

Для каждого значения параметра k параметр p будет пробегать все значения от 1 до a. Для каждого значения p и соответствующего значения k будет выполнен оператор вывода значений k и p. Будут выведены числа:

1,1, 1,2, ..., 1,a, 2,1, 2,2, ..., 2, a, 20,1, 20,2, ..., 20, a.

Примечание. Оператор цикла, как и условный оператор, нельзя помещать непосредственно после **then.**

Задачи.

1. Написать программу вычисления величин

$$a_1b_1 + a_2b_2 + \ldots + a_{80}b_{80} \text{ и } a_1b_{80} + a_2b_{79} + \ldots + a_{80}b_1.$$

2. Написать программу вычисления суммы

$$1 - \frac{x}{1!} + \frac{x^2}{2!} - \ldots + (-1)^n \frac{x^n}{n!}.$$

3. Даны массивы $a_1, a_2, \ldots, a_{300}$, $b_1, b_2, \ldots, b_{182}$. Написать программу построения массива $c_1, c_2, \ldots, c_{482}$, элементы которого соответственно равны $a_1, a_2, \ldots, a_{300}$, $b_1, b_2, \ldots, b_{182}$.

4. Написать программу выполнения следующего задания: выяснить, является ли данный массив a_1, a_2, ..., a_{550} упорядоченным по убыванию.

5. На плоскости даны 700 точек, эти точки попарно соединены отрезками. Написать программу вычисления длины наибольшего из отрезков. Считать, что координаты i-й точки суть (x_i, y_i) и массивы x_1, ..., x_{700}; y_1, ..., y_{700} заданы.

Полезно прорешать еще раз все задачи предыдущего параграфа, пользуясь операторами цикла.

* * *

Алгол 60 не единственный алгоритмический язык, которым пользуются программисты. Этот язык хорош прежде всего для решения вычислительных задач, т. е. для работы с данными числового типа (**real** и **integer**). Хотя числами и можно представлять весьма сложные структуры, для удобства программирования созданы и применяются языки, позволяющие работать с данными специальных типов (формулами, текстами и т. д.). Одна и та же вычислительная машина может понимать несколько языков: первоначально она владеет одним (входным) языком, но ее снабжают программами-переводчиками, которые будут переводить на этот язык программы с других языков. Об этом пойдет разговор в следующих главах.

ГЛАВА III

ВЫЧИСЛИТЕЛЬНАЯ МАШИНА И ЕЕ ВХОДНОЙ ЯЗЫК

Разные марки вычислительных машин имеют разные входные языки, из тех и других трудно выбрать наиболее типичные. Мы рассмотрим упрощенную модель вычислительной машины. Хотя и эта модель, и ее входной язык являются вымышленными, они довольно точно отражают то, чем на самом деле располагают программисты.

§ 1. Память машины. Машинные слова

В памяти машины хранится программа и все рассматриваемые в ходе вычислений величины. Память состоит из отдельных ячеек, в ячейке может храниться любая последовательность из 24 двоичных цифр — нулей и единиц. Каждую такую последовательность будем называть машинным словом (иногда — просто словом). Примеры машинных слов:

111111111111111111111111
010110110000011000000111

Если в ячейке хранится слово $\alpha_1\alpha_2 \ldots \alpha_{24}$, то цифра α_i $(1 \leqslant i \leqslant 24)$ называется содержимым i-го разряда ячейки.

Память нашей модели машины будет содержать 2^{15} ячеек, пронумерованных числами от 0 до $2^{15} - 1$. Номер ячейки называется ее адресом.

В арифметических задачах исходными данными служат числа. Число для машины — это слово специального вида. Если бы нужны были только целые числа, то можно было бы первый разряд ячейки использовать для записи знака числа (0 — это плюс, 1 — это минус), а в

остальных 23 разрядах записывать абсолютную величину числа в двоичной системе. Так удалось бы записывать в ячейке любое целое число с абсолютной величиной $\leqslant 2^{23}$. Но приходится работать и с нецелыми числами, поэтому выбран универсальный способ записи, основанный на представлении числа $a \neq 0$ в двоичной нормализованной форме $2^v \cdot m$, где v — целое число и $1/2 \leqslant |m| < 1$. Число v называется *двоичным порядком числа a*, число m — *двоичной мантиссой* этого числа. Соответствующее машинное слово устроено так: первый разряд — это знак числа, второй — знак порядка, следующие 7 разрядов — целое число, равное абсолютной величине порядка, последние 15 разрядов — цифры, стоящие после запятой в абсолютной величине мантиссы. Число 1 запишется в виде слова

000000001100000000000000,

число —2 в виде

100000010100000000000000.

Запишем число —1/3. Это число изображается бесконечной двоичной дробью —0,01010101 ..., учитываются первые цифры:

110000001101010101010101.

Число 0 записывается словом

000000000000000000000000.

Числа — не единственный встречающийся вид данных. При обработке текстов каждой литере (букве, цифре, знаку операции, знаку препинания и т. д.) присваивается номер (числовой код) — целое неотрицательное число, меньшее 256, и каждая ячейка хранит три литеры. Номер каждой литеры, записанный в двоичной системе, занимает 8 последовательных разрядов (если запись номера литеры в двоичной системе содержит менее 8 цифр, то старшие цифры объявляются нулями, например, 1 заменяется на 00000001).

Машинное слово может быть и *командой*, т. е. элементарной инструкцией машине. Машина может выполнять над словами некоторые операции, за каждой такой операцией закреплен код — целое число. Первые 9 разрядов команды заняты кодом операции, а последние 15 — адресом ячейки, в которой может, например, храниться слово, над которым выполняется операция. Если

код некоторой операции 1010, а адрес ячейки, который по смыслу надо указать в команде, — 10, то команда запишется так:

$$\underbrace{000001010}_{\text{код операции}}\ \underbrace{000000000000010}_{\text{адрес ячейки}},$$

Соглашение: 24-разрядные двоичные машинные слова слишком длинны. При записи машинного слова на бумаге его разбивают на грани по 3 цифры, рассматривают каждую грань как запись некоторого числа в двоичной системе и записывают это число одной из цифр 0, 1, 2, ... 7. Получается 8-разрядное восьмеричное слово. Приведенные в качестве примера в начале параграфа двоичные слова запишем так:

77777777, 26603007.

Числа 1, —2, —1/3, 0 представляются словами

00140000, 40240000, 60152525, 00000000.

Последний пример команды перепишем так:

$$\underbrace{012}_{\text{код операции}}\ \underbrace{00002}_{\text{адрес ячейки}}$$

Задачи.

1. Рассмотреть 24 буквы названия настоящей книги. Получить машинное слово, заменяя каждую букву, изображающую гласный звук, единицей, и каждую букву, изображающую согласный звук, нулем. Записать это двоичное слово в виде восьмеричного.

2. Записать в виде машинных слов числа 3,14; 100; —1,1; 8/17.

Какие числа записаны машинными словами 123456701, 400400400, 4444444?

3. Показать, что если двоичное слово является записью ненулевого числа, то его десятая цифра — единица.

4. Пусть буквы русского алфавита АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ пронумерованы числами от 11 до 43. Некоторое слово русского языка записано в виде двух машинных слов 03410023, 03001401 так, как предлагалось в тексте параграфа: в каждое слово входит 3 номера литер-букв. Прочитать слово.

5. Доказать, что для перевода целого числа из двоичной системы в восьмеричную достаточно разбить последовательность двоичных цифр числа (начиная с младшей цифры) на грани по 3 цифры и каждую грань перевести в число от 0 до 7. Например, для перевода числа сто, записанного в двоичной системе как 1100100, в восьмеричную систему можно поступить так: разбиваем его запись на грани 1 |100| 100 и получаем 144. Обратно, для перевода из восьмеричной системы в двоичную достаточно заменить каждую восьмеричную цифру тремя двоичными.

Сформулировать подобные правила для записей целых чисел в системах с основаниями q и q^k.

§ 2. Арифметические операции. Команды перехода

Машина может выполнять ряд арифметических операций над числами. Типичные арифметические операции имеют два операнда.

При выполнении операций используется специальная ячейка — так называемый *сумматор*. Эта ячейка не входит в число занумерованных ячеек, о которых шла речь выше. Сумматор не упоминается в командах явно. Его роль станет ясной после рассмотрения примеров.

Операция сложения имеет код 001 *). Выполнение команды 00101173 приведет к сложению числа из сумматора с числом из ячейки 01173; результат будет помещен опять в сумматор (прежнее содержимое сумматора пропадет). Аналогично выполняются операции вычитания с кодом 002, умножения с кодом 003, деления с кодом 004, возведения в степень с кодом 005: первый операнд берется из сумматора, второй — из ячейки, адрес которой указан в команде. Результат остается в сумматоре.

Имеются две операции считывания слов (в частности, чисел): 011 — считывание слова из сумматора в ячейку, адрес которой указан в команде; 012 — считывание слова из ячейки в сумматор. Считывание значения из ячейки или сумматора не приводит к изменению содержимого этой ячейки или сумматора.

*) В конце этой главы приведен список кодов всех операций рассматриваемой модели вычислительной машины; разбирая примеры и решая задачи, можно заглядывать в этот список.

Соглашение о ячейке 00000: в этой ячейке всегда хранится слово 00000000, ее содержимое ничем нельзя изменить.

Команды выполняются специальным устройством машины — *процессором.* Выполнив находящуюся в некоторой ячейке команду, процессор, как правило, переходит к выполнению команды из следующей ячейки. Этот порядок нарушается только специальными командами перехода и остановки, о них будет рассказано позже.

Напишем последовательность команд, выполнение которой приведет к вычислению значения $y = ax^2 + bx + c$ и помещению его в ячейку 40000, коль скоро значения a, b, c, x размещены в ячейках 40001, 40002, 40003, 40004.

Слева от команды указываем номер ячейки, в которой будет находиться эта команда. Справа будем давать пояснения, сумматор в них будет обозначаться σ. И левая, и правая части в память машины не попадают.

00001	01240001	$\sigma := a$
00002	00340004	$\sigma := \sigma \times x\ (= ax)$
00003	00140002	$\sigma := \sigma + b\ (= ax + b)$
00004	00340004	$\sigma := \sigma \times x\ (= ax^2 + bx)$
00005	00140003	$\sigma := \sigma + c\ (= ax^2 + bx + c)$
00006	01140000	$y := \sigma$

Если заставить процессор приступить к выполнению команды из ячейки 00001, то будет выполнена вся выписанная последовательность.

Приведенная последовательность команд, в принципе, могла бы быть размещена в любых идущих подряд ячейках, лишь бы она не попадала в «числовые» ячейки 40000, 40001, ..., 40004.

Во входном языке машины отсутствуют аналоги алгольных условных операторов и операторов цикла. Здесь все строится на командах перехода. Рассмотрим две из них. Первая — команда безусловного перехода; после кода операции 021 указывается адрес команды, к выполнению которой надо перейти. Вторая (команда условного перехода) имеет код операции 022. По этой команде процессором исследуется содержимое сумматора, которое для применения этой команды обязано быть числом. Если это число $\geqslant 0$, то происходит переход к команде с адресом, указанным в команде

перехода. Если число < 0, то выполняется следующая команда.

Рассмотрим последовательность команд вычисления $y = |x|$. Пусть x расположен в ячейке 01000, а y должен оказаться в ячейке 01001:

00010	01201000	$\sigma := x$
00011	02200014	если $\sigma \geqslant 0$, то на 00014
00012	01200000	$\sigma := 0$
00013	00201000	$\sigma := \sigma - x (= -x)$
00014	01101001	$y := \sigma$

Эта последовательность команд, в отличие от предыдущей, не допускает перемещения в памяти из-за команды условного перехода, которая отсылает к ячейке с явно выписанным адресом 00014.

Задачи.

1. Написать последовательность команд вычисления $y = \left(\frac{ax+b}{cx+d}\right)^n$, где a, b, c, d, x, n — числа, расположенные, начиная с ячейки 50000. Значение y должно занять место прежнего значения x.

2. Написать последовательность команд обмена значениями между ячейками 10000 и 10001.

3. Написать последовательность команд вычисления среднего арифметического и среднего геометрического положительных чисел a и b, значения которых находятся в ячейках 74375 и 74376. Считать, что в ячейке 74377 находится значение 1/2. Результаты вычисления должны быть помещены в ячейки 74400 и 74401.

4. Написать последовательность команд вычисления функции $\operatorname{sign} x$:

$$\operatorname{sign} x = \begin{cases} 1, & \text{если } x > 0, \\ 0, & \text{если } x = 0, \\ -1, & \text{если } x < 0. \end{cases}$$

Считать, что значение x находится в ячейке 10000, а значение $\operatorname{sign} x$ должно быть в сумматоре.

5. Написать последовательность команд вычисления суммы $x + x^2 + x^4 + x^8 + \ldots$, учитывая только слагаемые, бо́льшие положительного числа a. Данные числа a и x ($|x| < 1$) расположены в ячейках 70001 и 70002. Результат получить в ячейке 70000.

§ 3. Выполнение программы

Чтобы попасть в память машины, программа должна пройти через устройство ввода, а для этого надо ее сначала перенести с листа бумаги, на котором она записана, на перфокарты. Перфокарта — это особого формата листок тонкого картона. В определенных местах перфокарты могут быть пробиты отверстия. Отверстие рассматривается как представление цифры 1, а отсутствие отверстия — цифры 0, это позволяет комбинациями отверстий записывать двоичные слова. Перфокарта может вместить несколько таких слов.

Группа всех слов, которые должны быть в конечном счете введены в память, разбиваются на порции. Каждая порция вводится в один прием и размещается в последовательных ячейках памяти. Сейчас будет сказано об управлении этим вводом.

Каждая порция двоичных слов начинается с *заголовка*. Заголовок сам является словом; в дальнейшем считаем, что он всегда занимает отдельную перфокарту. Структура заголовка $000\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5$, где $\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5$ — запись в восьмеричной системе числа машинных слов в порции. Если в порции 19 слов, то ее заголовком будет 00000023. Заголовок сам не попадет в память, при каждом обращении к устройству ввода он считывается в сумматор и исследуется процессором.

После запуска машины процессор первым делом включает устройство ввода. Прочитав заголовок самой первой порции, процессор вводит столько слов, сколько указано в заголовке (т. е. всю первую порцию слов). Эти слова размещаются в памяти, начиная с ячейки 00001. После этого устройство ввода выключается, процессор считывает слово из ячейки 00001 и, рассматривая его как первую команду, приступает к выполнению программы.

Все остальные порции вводятся по команде ввода: код операции 031, за ним указывается адрес ячейки, начиная с которой надо ввести очередную порцию слов. Для составления простейших программ необходимы еще две команды — вывода результатов и остановки.

По команде с кодом операции 032 устройство вывода печатает на бумаге содержимое нескольких последовательных ячеек. В адресной части этой команды помещается адрес первой из ячеек; в сумматоре должно

находиться слово, имеющее вид заголовка и указывающее количество ячеек. Если в сумматоре находится слово 00000001, то выполнение команды 03201000 приведет к выводу слова из ячейки 01000. По команде с кодом операции 041 и произвольным адресом выполнение программы заканчивается.

Вернемся к задаче вычисления значения $y = ax^2 + bx + c$. Эта программа будет размещена, начиная с ячейки 00001, а для величин y, a, b, c, x будут отведены, как и прежде, ячейки с 40000 по 40004. Используем имеющуюся заготовку.

σ	00000013	заголовок
00001	03140001	ввод (a, b, c, x)
00002	01240001	как прежде
00003	00340004	
00004	00140002	
00005	00340004	
00006	00140003	
00007	01140000	
00010	01100013	подготовка заголовка
00011	03240000	вывод (y)
00012	04100000	конец
00013	00000001	слово, используемое командой 00010

σ	00000004	заголовок
40001	. . . .	a
40002	. . . .	b
40003	. . . .	c
40004	. . . .	x

Во второй порции помещаются записи значений a, b, c, x. Эта порция не входит в программу. Программа может быть выполнена для любых значений величин a, b, c, x; в этом ее универсальность.

Слово в ячейке 00013 — не команда, а константа, необходимая для команды вывода, но оно включается в программу. Команда остановки препятствует интерпретации этого слова как команды.

Вторая из рассмотренных в предыдущем параграфе последовательностей команд при превращении в законченную программу доставляет больше хлопот: если все сделать, как при вычислении значения трехчлена, то по-

следовательность команд введется, начиная с ячейки 00001, а это не годится, так как последовательность неперемещаема в памяти и непременно должна располагаться, начиная с ячейки 00010.

На первый взгляд, эту ситуацию можно не рассматривать: всегда программу можно писать в предположении, что она будет располагаться, начиная с ячейки 00001. Но программы иногда составляют из отдельных частей уже имеющихся программ.

В нашем случае можно поступить так:

σ	00000002	заголовок
00001	03100007	ввод команд, начиная с ячейки 00007
00002	02100007	переход на 00007
σ	00000012	заголовок
00007	03101000	ввод (x)
00010	01201000	} как прежде
00011	02200014	
00012	01200000	
00013	00201000	
00014	01101001	
00015	01100020	подготовка заголовка
00016	03201001	вывод (x)
00017	04100000	конец
00020	00000001	слово, используемое командой 00015
σ	00000001	
01000	. . . .	x

Первоначально будет выполнена часть программы, обеспечивающая пуск; ее задача — заставить процессор ввести основную часть программы и перейти к ее выполнению. Пусковая часть содержит две команды, они будут введены по общему правилу в ячейки 00001 и 00002.

Примечание. Числовые исходные данные и результаты имеют вид мащинных слов. Обычно входной язык машины содержит команды, позволяющие вводить десятичные числа и переводить их в двоичную нормализованную форму, и команды вывода результата в десятичном виде. Десятичная запись числа

рассматривается как текст и может быть представлена, например, способом, показанным в § 1. Мы здесь этими вопросами заниматься не будем.

Задачи.

1. Написать программу вычисления $\operatorname{sign} x$ (см. задачу 4 предыдущего параграфа).

2. Даны два числа a, b. Написать программу, в результате выполнения которой эти числа выводятся в обратном порядке: b, a.

3. Переделать программу вычисления квадратного трехчлена, включив в пусковую часть программы команду ввода значений a, b, c, x.

4. Скомпоновать из имеющихся последовательностей команд вычисления $|x|$ и ax^2+bx+c программу вычисления $a|x|^2+b|x|+c$.

Указание. Последовательность команд вычисления $|x|$ дополнить командой безусловного перехода.

5. Написать программу вычисления среднего арифметического и среднего геометрического двух данных положительных чисел a и b. Машинное слово, изображающее необходимое для вычислений число 1/2, поместить вместе с командами программы (команды перехода и остановки должны предотвратить истолкование процессором этого слова как команды).

§ 4. Регистры. Обработка массивов

Рассмотрим задачу вычисления суммы n чисел $a_1, a_2, \ldots, a_n$, расположенных в последовательных ячейках памяти. Для прибавления к накопленной в сумматоре сумме первых $i-1$ слагаемых очередного слагаемого a_i достаточно выполнить команду 001 (a_i), где (a_i) — адрес ячейки, хранящей a_i. Если n известно уже при составлении программы, то можно выписать явно все необходимые команды. Если же n будет определяться только при выполнении программы, то это невозможно.

В таких случаях обращаются к *регистрам*. Регистры — это специальные ячейки; их в машине 7, и они пронумерованы числами 1, 2, 3, 4, 5, 6, 7. Каждый регистр способен вместить произвольное 15-разрядное двоичное слово, которое можно записывать как 5-раз-

рядное восьмеричное. Назначение регистров — изменение адресной части команд. Предложенные в предыдущих параграфах операции имели трехзначные восьмеричные коды, но первой из трех всегда была цифра 0. Вместо первого нуля можно указать номер регистра. Например, можно использовать команду 20150000: к содержимому сумматора будет прибавлено число из ячейки с адресом, равным сумме 50000 и содержимого второго регистра. Пусть во втором регистре находится слово 00003, тогда к содержимому сумматора будет прибавлено содержимое ячейки 50003. Итак, адресная часть команды изменяется на величину содержимого, указанного в первом восьмеричном разряде кода операции регистра. При сложении адресной части команды с содержимым регистра может получиться 16-значное двоичное число. В таком случае старшая единица отбрасывается.

В программу вычисления $a_1 + \ldots + a_n$ вместо команд $001(a_i)$, $i = 1, 2, \ldots, n$, достаточно вставить цикл с командой $201(a_1)$. Содержимое второго регистра должно при выполнении этого цикла принимать значения 00000, 00001, 00002, ... (n значений).

Чтобы составлять такие программы, нужны операции, позволяющие изменять и исследовать содержимое регистров.

Операция с кодом 013: 15 последних двоичных разрядов содержимого сумматора помещаются в регистр, номер которого указан в адресной части команды. Если в сумматоре содержится слово 67125530, то в результате выполнения команды 01300001 в первый регистр попадет слово 25530.

Операция с кодом 014: содержимое регистра с указанным в адресной части команды номером заносится в последние разряды сумматора, первые разряды сумматора заполнятся нулями.

Слова произвольного вида могут изменяться, в частности, с помощью операции с кодом 051: содержимое сумматора и ячейки, указанной в адресной части команды, складываются как 24-разрядные двоичные числа. Если в результате сложения возникает 25-разрядное число, то самая старшая единица пропадает. Пусть в ячейке 10005 хранится слово

111011011000010110100000,

или, в восьмеричном виде, 73541320; пусть содержимое сумматора представляет собой

011011011100001010010001,

или, в восьмеричном виде, 33341221. Поскольку

$$\begin{array}{r} 111011101100001011010000 \\ +\ 011011011100001010010001 \\ \hline 101011100100001010110000 1 \end{array}$$

то в результате восполнения команды 05110005 содержимым сумматора станет слово

010111001000010101100001,

или, в восьмеричном виде, 27102541.

Для сравнения двух слов будет использоваться операция поразрядного сравнения, ее код 052. Слово, хранящееся в ячейке, которая указана в адресной части команды, сравнивается поразрядно с содержимым сумматора.

Если содержимое разрядов совпало, то в этом же разряде слова-результата помещается 0, иначе 1. Пусть содержимое ячейки 10005 и сумматора такие же, как в предыдущем примере; тогда в результате выполнения команды 053100005 в сумматоре появится слово

100000110000000001000001,

или, в восьмеричном виде, 40600101.

Учесть результат сравнения двух слов можно с помощью операции условного перехода с кодом 023: переход к команде с адресом, записанным после кода 023, если содержимым сумматора является ненулевое слово.

Теперь у нас есть все необходимое, чтобы составить программу вычисления $s = a_1 + \ldots + a_n$. Числа $a_1, \ldots, a_n$ будут расположены, начиная с ячейки 10000, число n — в ячейке 50000, для s отводится ячейка 50001. В комментариях к этой и следующим программам содержимое регистров будет обозначаться, соответственно, $\rho_1, \ldots, \rho_7$.

Программа:

σ	00000022	заголовок
00001	03110000	ввод $(a_1, \ldots, a_n)$
00002	01150000	$n := \sigma$
00003	01210000	$s := a_1$
00004	01150001	
00005	01200022	$\rho_2 := 1$
00006	01300002	
00007	01250001	$\sigma := s (= a_1 + \ldots + a_{i-1})$
00010	20110000	$\sigma := \sigma + a_i (= a_1 + \ldots + a_i)$
00011	01150001	$s := \sigma$
00012	01400002	$\rho_2 := \rho_2 + 1$
00013	05100021	
00014	01300002	
00015	05250000	если $n \neq \sigma$, то на 00007
00016	02300007	
00017	01100022	подготовка заголовка
00020	03250000	вывод (s)
00021	04100000	конец
00022	00000001	слово, используемое командами 00005, 00017
	. . . .	заголовок и числа $a_1, \ldots, a_n$
	. . . .	
	. . . .	
	. . . .	
	. . . .	

Циклически выполняются команды 00007—00016, команда 00010 использует второй регистр. При составлении этой программы учтено, что заголовок порции при вводе в машину попадает в сумматор (команда 00002).

Задачи.

1. Написать программу вычисления суммы положительных и числа отрицательных элементов массива $a_1, \ldots, a_n$.

2. Написать программу, при выполнении которой выясняется, является ли данный массив $a_1, \ldots, a_n$ неубывающим, т. е. верно ли, что $a_1 \leqslant a_2 \leqslant \ldots \leqslant a_n$. Ответом должно быть число 1 или 0.

3. Написать программу вычисления суммы $a_1 - 2a_2 + 3a_3 - \ldots \pm na_n$, где знак перед последним слагаемым определяется четностью n.

4. Как уменьшить содержимое некоторого регистра на 1?

5. Используя решение задачи 4, написать программу вычисления величины $a_1b_n + a_2b_{n-1} + \ldots + a_nb_1$ для данных массивов $a_1, \ldots, a_n$ и $b_1, \ldots, b_n$.

§ 5. Операции над словами произвольного вида

В предыдущих параграфах мы занимались главным образом арифметическими задачами, хотя не менее важны и интересны задачи обработки нечисловых данных, например упоминавшиеся задачи обработки текстов.

Возможны и программы, предназначенные для построения или анализа других программ. Рассмотрим следующий простой пример. Даны два массива машинных слов. Каждое слово является командой. Требуется ввести массивы в память машины, начиная с указанных ячеек, и заставить машину выполнить программы одну за другой. Предположим, что программы и ячейки, используемые в них, не налагаются друг на друга. Тогда для решения задачи достаточно заменить в первой программе все команды остановки на команды безусловного перехода к началу второй программы.

Кроме описанных операций двоичного сложения и сравнения слов, мы будем использовать операции сдвига слова на несколько двоичных разрядов влево и вправо.

Сдвиг влево: код операции 053, после кода указывается число, на которое надо сдвинуть влево содержимое сумматора (существенный момент: после кода операции стоит не адрес, а само число). Пусть, например, сумматор содержит слово

011011011100001010010001,

или, в восьмеричной записи, 33341221. Выполнение команды 05300023 приведет к появлению в сумматоре слова

100010000000000000000000

5 последних цифр прежнего слова

или, в восьмеричной записи, 42000000 (так как 23 — это запись в восьмеричной системе числа 19).

Операция с кодом 054 — сдвиг вправо. Выполнение команды 05400023 привело бы к появлению в сумматоре

слова

$$\underbrace{00000000000000000001101}_{\substack{\text{5 первых}\\ \text{цифр}\\ \text{прежнего}\\ \text{слова}}}$$

или, в восьмеричной записи, 00000013.

Теперь вернемся к примеру этого параграфа и напишем требуемую программу. Пусть заданные массивы программы надо ввести в память, начиная с ячеек 30000 и 40000. Число слов в первом массиве в комментариях обозначим через n.

σ	00000025	заголовок
00001	03130000	ввод 1-го массива
00002	01100100	$n := \sigma$
00003	03140000	ввод 2-го массива
00004	01200000	$\rho_1 := 0$
00005	01300001	
00006	01400001	$\rho_1 := \rho_1 + 1$
00007	05100023	
00010	01300001	
00011	11227777	$\sigma :=$ слово из 1-го массива
00012	05400020	совпадает ли код операции с 041?
00013	05200024	
00014	02300017	если не совпадает, то на 00017
00015	01200025	замена команды остановки
00016	11127777	
00017	01400001	$\sigma := \rho_1$
00020	05200100	сравнение ρ_1 с n
00021	02300006	если не совпало, то на 00006
00022	02103000	переход к выполнению 1-й программы
00023	00000001	слово, используемое командой 00007
00024	00000041	слово, используемое командой 00013
00025	02104000	слово, используемое командой 00015
	. . . .	заголовок и 1-й массив слов
	. . . .	
	. . . .	
	. . . .	заголовок и 2-й массив слов
	. . . .	

Операции над словами произвольного вида важны для реализации на машине целочисленной арифметики. Выполняя арифметические операции над числами в нормализованной форме, машина не разбирает, целые ли попались операнды, и, возможно, будет вычислять результат приближенно. С последним обстоятельством, как известно, не всегда можно примириться.

Задачи.

1. Написать программу, после выполнения которой выясняется, совпадают ли двоичные порядки двух данных чисел. Ответом должно быть число 1 или 0.

2. Дан массив машинных слов, изображающий некоторый текст: каждое слово содержит код трех литер. Пусть двоичный код буквы А есть 0001011. Написать программу проверки, входит ли в данный текст буква А. Ответом должно быть число 1 или 0.

3. Обратным кодом двоичного слова называется слово, получающееся из исходного заменой 1 на 0 и 0 на 1. Пусть слова a и b рассматриваются как целые числа $<2^{24}$. Пусть в этом смысле $a \geqslant b$. Доказать, что для построения слова $a - b$ достаточно применить операцию сложения 051 к словам a и обратному коду слова $b - 1$ (сравните с задачей 4 § 4).

4. Написать программу преобразования неотрицательного целого числа $<2^{24}$ из нормализованной формы в слово, являющееся записью данного числа в двоичной системе. Старшие незанятые разряды, как обычно, заполняются нулями.

Указание. Показать, что если p — двоичный порядок данного числа, то для решения исходной задачи достаточно сдвинуть число на 9 разрядов влево, а результат этого сдвига — на $24 - p$ разрядов вправо; $24 - p = 25 - p - 1 = 25 - (p + 1)$ (далее см. предыдущую задачу). Значение $24 - p$ заносится в какой-нибудь регистр, а в команде сдвига вправо используется номер этого регистра.

5. Написать программу вычисления a^n (n — целое неотрицательное, a — любое число) двоичным алгоритмом (см. гл. I, § 4).

Указание. Деления на 2 с остатком не понадобятся, так как число n записано в двоичной системе и можно читать его цифры по очереди.

Разобранные в этой главе примеры показывают, что составление программ решения даже несложных задач на входном языке вычислительной машины — дело трудоемкое и малоприятное. Это и явилось стимулом к созданию богатых языков программирования вроде алгола 60 и к разработке методов перевода (трансляции) программ на этих языках в программы на входном языке.

Вопросы, связанные с трансляцией, обсуждаются в следующей главе.

В заключение приведем список операций рассмотренной модели вычислительной машины. Этот список может пригодиться для справок. В скобках указаны страницы книги, на которых операции разъясняются подробно.

Арифметические

001 — сложение (46),
002 — вычитание (46),
003 — умножение (46),
004 — деление (46),
005 — возведение в степень (46).

Считывания

011 — из сумматора в ячейку (46),
012 — из ячейки в сумматор (46),
013 — содержимого 15 младших разрядов сумматора в регистр (53),
014 — из регистра в младшие разряды сумматора, старшие 9 разрядов заполняются нулями (53).

Перехода

021 — безусловный (47),
022 — при неотрицательном содержимом сумматора (47),
023 — при наличии ненулевого слова в сумматоре (54).

Ввода и вывода

031 — ввод порции слов (49),
032 — вывод порции слов (49).

Остановки

041 (50).

Над словами произвольного вида

051 — двоичное сложение; если возникает 25-значное число, то самая старшая единица пропадает (53),
052 — поразрядное сравнение (54),
053 — сдвиг вправо (56),
054 — сдвиг влево (56).

ГЛАВА IV

ТРАНСЛЯЦИЯ

В этой главе в самых общих чертах излагается один из методов *трансляции* (перевода) алгольных программ на язык машины.

Изложение носит общий характер, в разбираемых же иллюстративных примерах под языком машины подразумевается язык модели машины, обсуждавшейся в предыдущей главе.

§ 1. Преобразование последовательностей символов

Рассмотрим пока некоторые простые преобразования последовательностей символов, они не решают задачи трансляции, но оказываются нужными при ее решении.

Поэлементное преобразование последовательности символов состоит в замене по определенному правилу каждого символа. Это правило иногда задается таблицей. Простейший способ кодирования текста — замена каждой буквы табличным кодом. Все переменные в программе могут при трансляции заменяться по таблице адресами ячеек, хранящих их значения, — такая таблица создается при анализе описаний программы.

Второе преобразование, рассматриваемое здесь, — перестановка элементов. Перестановки при трансляции получаются главным образом с помощью *магазина*. Магазин задается упорядоченный совокупностью элементов, которые в наших рассмотрениях будут машинными словами или порциями машинных слов. Принцип устройства магазина — «последним вошел — первым вышел», в названии подразумевается сходство с магазином огнестрельного оружия. Часто при объяснении принципа перестановки элементов с помощью магазина прибегают

к аналогии не с оружием, а с железнодорожным тупиком (рис. 8).

Предполагаются две операции: занесение очередного символа исходной последовательности в магазин (обозначим эту операцию В) и вывод из магазина последнего из попавших туда символов, про который говорят, что он находится на *верхушке* магазина (эту операцию обозначим ИЗ). Выведенный символ присоединяется в конец последовательности - результата. Последовательность a_1, a_2 с помощью операций В, В, ИЗ, ИЗ преобразуется в a_2, a_1. Не любую перестановку элементов последовательности можно получить с помощью операций В и ИЗ. Например, из a_1, a_2, a_3 никак не получить a_3, a_1, a_2: когда a_3 выходит из магазина, a_1 и a_2 находятся там, но a_1 попал в магазин раньше, чем a_2, и обязан выйти позже.

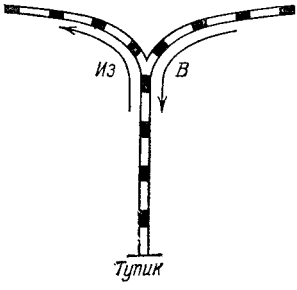

Рис. 8.

Обсудим вкратце представление таблиц и магазина в памяти машины. Таблица замены одного машинного слова другим может иметь вид двух массивов: преобразуемое слово разыскивается среди слов первого массива, из второго массива берется слово, имеющее тот же порядковый номер, что и найденное в первом массиве. Порядок слов в первом массиве выбирают таким, чтобы поиск был возможно более быстрым; чаще всего этот порядок аналогичен словарному.

Магазин занимает несколько последовательных ячеек, в которые попадают переставляемые элементы; еще одна ячейка (или регистр) хранит адрес верхушки магазина, эта ячейка, или регистр, называется указателем верхушки. Если упомянутые последовательные ячейки имеют адреса m, $m+1$, $m+2$, ... и в некоторый момент магазин содержит k слов, то эти слова находятся в ячейках с адресами m, $m+1$, ..., $m+k-1$ и в указателе верхушки находится адрес $m+k-1$. Если надо поместить в магазин еще один элемент, то он должен попасть в ячейку с адресом $m+k$, соответственно должен увеличиться адрес, хранящийся в указателе верхушки. Извлечение элемента из магазина приводит к

извлечению элемента из ячейки, адрес которой хранится в указателе верхушки; содержимое указателя верхушки уменьшается после этого на 1.

Если магазин должен оперировать с элементами, представляющими собой порции слов, то в начало каждой порции добавляется слово-заголовок, указывающее количество слов в элементе. Содержимое указателя верхушки изменяется при операциях занесения и вывода элемента уже не на единицу: если в заголовке указано число l, то содержимое указателя верхушки должно быть изменено на $l+1$ — слагаемое 1 добавлено потому, что заголовок тоже занимает ячейку.

Задачи.

1. Последовательность букв названия этой книги заменить последовательностью чисел. Число, сопоставляемое букве, определяется как количество различных букв, предшествующих первому вхождению рассматриваемой буквы в название.

2. Последовательность чисел 1, 2, 3, 4 преобразуется операциями В, В, В, ИЗ, ИЗ, В, ИЗ, ИЗ. Выписать получающуюся перестановку.

3. Выписать перестановки элементов a_1, a_2, a_3, которые могут быть получены с помощью операций В и ИЗ.

4. Пусть таблица имеет вид двух массивов из n элементов и элементами первого массива являются числа, упорядоченные по возрастанию. Продумать метод поиска номера данного числа в первом массиве, основанный на делении массива на две части: сравнением с элементом с номером $[n/2]$ выясняется, в какой половине массива надо продолжить поиск. Показать, что число необходимых сравнений не превзойдет $[\log_2 n]+1$.

5. Пусть указателем верхушки магазина служит регистр 5. Написать команды вывода элемента в сумматор и занесения из сумматора в магазин. Для вспомогательных целей можно использовать ячейку 70000. Какие машинные слова необходимо запасти для преобразования указателя верхушки? Считать, что они запасены в последовательных ячейках, начиная с ячейки 70001.

§ 2. Вычисление значения выражения

Вычисление значения выражения требует соблюдения старшинства операций. Например, применение известного из школьного курса математики алгоритма к

выражению $b \uparrow 2 - 4 \times a \times c$ включает предварительное установление порядка выполнения операций

$$b \overset{1}{\uparrow} 2 \overset{4}{-} 4 \overset{2}{\times} a \overset{3}{\times} c;$$

для этого выражение просматривается несколько раз. Выполнение каждой операции дает некоторое число — его приходится записывать отдельно от выражения, запоминая тот фрагмент выражения, для которого число является значением. Соответственно до выполнения операции должно быть установлено, к значениям каких фрагментов исходного выражения она должна применяться. Так, перед выполнением операции вычитания, помеченной цифрой 4, устанавливается, что применяется к значениям выражений $b \uparrow 2$ и $4 \times a \times c$.

Ясно, что полное описание этого алгоритма было бы очень сложным, выполнение же такого алгоритма на машине потребовало бы и больших затрат времени (которое будет уходить главным образом не на выполнение операций, а на просмотры выражения, поиски нужных промежуточных результатов и т. д.), и большого числа ячеек памяти (например, потребуется много ячеек памяти для сопоставления промежуточным результатам фрагментов исходного выражения).

Сейчас будет предложен экономный алгоритм вычисления значения выражения, использующий 2 магазина для перестановки элементов выражения с учетом старшинства операций и для хранения промежуточных результатов.

Магазины обозначим M_1 и M_2; в M_1 будут попадать знаки операций, в M_2 — числа, участвующие в записи выражения, значения переменных и все промежуточные числовые значения.

Ограничимся выражениями, состоящими только из чисел и переменных без индекса, связанных знаками операций $\uparrow$, $\times$, $/$, $+$, $-$. Знак минус всегда будет знаком двухместной операции вычитания, выражения вроде $-a + 1$ исключаются из рассмотрения. От этих ограничений можно было бы и отказаться, но это удлинило бы изложение. Пока предположим также, что в выражении нет скобок.

Опишем алгоритм вычисления. Исходное выражение читается слева направо; если прочитано число, то оно заносится в M_2, если переменная — в M_2 заносится ее

значение; если же прочитан знак операции, то необходимо различать несколько случаев.

1) M_1 пуст; прочитанный знак помещается на верхушку M_1.

2) То же самое, если прочитанный знак обозначает операцию, которая старше и поэтому должна выполняться до операции, знак которой расположен на верхушке M_1.

3) Если операции равноправны или если та, знак которой только что прочитан в выражении, должна выполняться позднее, необходимо применить операцию, знак которой расположен на верхушке M_1, к двум верхним числам из M_2 (число на верхушке — второй операнд, число под ним — первый); знак операции на верхушке M_1 удаляется из M_1, вместо двух верхних чисел из M_2 помещается результат выполненной над ними операции.

В некоторый момент в исходном выражении не остается символов. Если пуст и M_1, то вычисление окончено, результат находится в M_2; иначе знаки операции извлекаются по очереди из M_1 и соответствующие операции применяются к числам из M_2.

Рассмотрим вычисление выражения $b \uparrow 2 - 4 \times a \times c$; значения переменных a, b, c обозначим A, B, C.

M_2	M_1	$2 - 4 \times a \times c$
B	$\uparrow$	
M_2	M_1	$-4 \times a \times c$
2	$\uparrow$	
B		
M_2	M_1	$-4 \times a \times c$
$B \uparrow 2$		
M_2	M_1	$\times a \times c$
4	$-$	
$B \uparrow 2$		
M_2	M_1	$\times c$
A	$\times$	
4	$-$	
$B \uparrow 2$		

M_2	M_1	$\times c$
$4 \times A$	—	
$B \uparrow 2$		

M_2	M_1
C	$\times$
$4 \times A$	—
$B \uparrow 2$	

M_2	M_1
$4 \times A \times C$	—
$B \uparrow 2$	

M_2	M_1
$B \uparrow 2 - 4 \times A \times C$	

Про знак операции говорят, что он имеет более высокий приоритет в сравнении с другим знаком, если обозначаемая им операция старше. В соответствующих случаях говорят о равных приоритетах и более низком приоритете. Рассмотренные знаки операций распадаются на группы равных по приоритету:

$\uparrow$
$\times$, /
$+$, $-$

Группы упорядочены по убыванию приоритета.

Полное доказательство, что предложенный алгоритм действительно приводит к вычислению значения выражения, основывается на возможности представления исходного выражения E в виде $E_1\theta_1E_2\theta_2 \ldots \theta_{k-1}E_k$, где $\theta_1, \theta_2, \ldots, \theta_{k-1}$ — знаки операций одинакового приоритета, а $E_1, E_2, \ldots, E_k$ — выражения, в которые не входят знаки операций с приоритетами, более высокими или равными приоритету знаков $\theta_1, \theta_2, \ldots, \theta_{k-1}$. Используя это, можно провести индукцию по числу групп равных по приоритету знаков операций — для каждого E_i, $i = 1, 2, \ldots, k$, это число по крайней мере на 1 меньше, чем для E.

Теперь дадим правило работы со скобками. Левая скобка заносится в M_1 сразу после прочтения. Прочтение правой скобки влечет выполнение всех операций,

знаки которых находятся в M_1 выше левой скобки; после выполнения этих операций обе скобки уничтожаются. Вот что будет происходить при вычислении $(a+b)\times c$:

M_2	M_1	$+b)\times c$
A	(	
M_2	M_1	$)\times c$
B	$+$	
A	(	
M_2	M_1	$)\times c$
$A+B$	(	
M_2	M_1	$\times c$
$A+B$		
M_2	M_1	
C	$\times$	
$A+B$		
M_2	M_1	
$(A+B)\times C$		

Задачи.

1. С помощью предложенного алгоритма вычислить значение $4\times 3/2-(2+2\times(3+4+5))$.

2. В некоторый момент вычисления значения выражения находящийся на верхушке M_1 знак не совпадает со вторым (т. е. следующим за ним в магазине) знаком, но совпадает с третьим знаком. Какие возможности существуют для верхних трех знаков из M_1?

3. Среди выражений $a\times b\times c$, $a\times(b\times c)$, $(a\times b)\times c$ найти такие, при вычислении значений которых с помощью предложенного алгоритма порядок выполнения операций будет одинаковым.

4. В выражении $a\times x\times y+b\times x\times z+c\times y\times z$ расставить скобки так, чтобы они однозначно определяли тот порядок выполнения операций, который возникает при вычислении исходного выражения по предложенному алгоритму.

5. Напомним, что отношение в алголе — два выражения, соединенных одним из знаков отношения $=$, $\neq$, $>$, $<$, $\geqslant$, $\leqslant$. Значением отношения будем считать 1,

если оно выполнено, и 0 — в противном случае. Вставить в приведенный в конце параграфа список групп знаков операций группу знаков отношений так, чтобы описанный метод вычисления значений стал пригодным для вычисления значений отношений.

§ 3. Трансляция выражений

Предложенный в § 2 алгоритм позволял несложными действиями с двумя магазинами получать значение выражения — число. Алгоритм можно преобразовать так, чтобы получалось не само число, а последовательность машинных команд вычисления этого числа.

В магазин M_2 теперь будут заноситься не числа, а порции из нескольких машинных слов, содержащие команды вычисления некоторого числа с помещением его в сумматор. В моменты, когда прежний алгоритм предписывал произвести операцию над двумя верхними элементами M_2, мы будем соединять две верхние порции команд в одну — выполнение получившихся команд должно привести к появлению в сумматоре соответствующего числа.

Оговоримся, что соединение нескольких порций команд в одну понимается не как простое приписывание порций друг к другу (такую операцию мы будем называть *склеиванием*), между исходными порциями команд вставляются некоторые связующие команды. И при соединении, и при склеивании заголовки исходных порций убираются, а в начало порции-результата помещается новый заголовок.

Иногда заносимая в M_2 порция будет содержать лишь одно слово: пусть в рассматриваемом выражении очередным элементом является переменная — прежний алгоритм предписывал занести в M_2 значение этой переменной, теперь же в M_2 пойдет слово, в младших разрядах которого записан адрес ячейки, хранящей значение переменной, а в старших разрядах — нули. Такое слово-заготовка легко превращается и в команду перенесения значения в сумматор, и в любую команду выполнения действия над содержимым сумматора и содержимым ячейки.

При вычислении значения выражения возникают некоторые промежуточные значения. Для них должны быть зарезервированы ячейки. Адреса этих ячеек будут участвовать в командах, возникающих в M_2.

Стандартный прием соединения двух порций из M_2 в одну при выходе из M_1 знака операции будет состоять в следующем. Обе порции выводятся из магазина, затем исследуется вторая порция, которая должна определять первый операнд. Если она состоит более чем из одного слова и, следовательно, является последовательностью команд получения первого операнда в сумматоре, то к ней добавляется команда считывания значения сумматора в одну из зарезервированных ячеек. Если вторая порция — из одного слова, которое, следовательно, является заготовкой команды, то последовательность команд получения первого операнда в некоторой ячейке нам не нужна. Таким образом, описанный первый этап даст последовательность команд, которая может оказаться пустой. Второй этап заключается в построении последовательности команд получения в некоторой ячейке второго операнда — если первая порция состоит из слова-заготовки, то эта последовательность оказывается пустой. Третий этап заключается в построении двух команд — считывания первого операнда в сумматор и выполнения нужной операции над содержимым сумматора и содержимым ячейки, выделенной для второго операнда. Три последовательности команд, получившиеся на трех этапах, склеиваются в одну и помещаются на верхушку M_2.

Посмотрим теперь, как возникают с помощью этого варианта алгоритма команды вычисления $(a+b)\times c$. Предполагается, что значения a, b, c хранятся в ячейках 01001, 01002, 01003. Каждая порция слов в M_2 предваряется заголовком, указывающим число слов в порции.

M_2	M_1	$+b)\times c$
00000001	(	
00001001		

M_2	M_1	$)\times c$
00000001	+	
00001002	(	
00000001		
00001001		

M_2	M_1	$)\times c$
00000002	(	
01201001		
00101002		

M_2	M_1	$\times c$
00000002		
01201001		
00101002		

M_2	M_1
00000001	$\times$
00001003	
00000002	
01201001	
00101002	

M_2	M_1
00000005	
01201001	
00101002	
01177777	
01277777	
00301003	

В заключение несколько замечаний.

Числа, встречающиеся в рассматриваемом выражении, как и переменные, должны заменяться словами-заготовками, содержащими адреса ячеек, хранящих эти числа.

В магазин M_1 попадают знаки арифметических операций, представленные некоторыми машинными словами. Естественно брать слова, в старших разрядах которых помещен код соответствующей операции. При этом станет удобным изготовление команд из слов-заготовок.

Продумывая последний алгоритм, мы игнорировали принятое в алголе разделение переменных и чисел по типам.

Задачи.

1. Сколько получится команд в результате трансляции выражения $a \times b \times c \times d$?

2. Оттранслировать выражение $a + b \times c$.

3. Программа, полученная в конце параграфа, после трансляции выражения $(a + b) \times c$ обладает очевидным дефектом — третью и четвертую команды можно удалить. Внести поправку в приведенную рекомендацию по соединению двух порций из M_2 в одну при выходе из M_1 знака операции.

4. Трансляция выражения $(a+b)\times c$ может дать последовательность команд, не включающих адреса зарезервированных ячеек, — надо воспользоваться решением задачи 3. Показать, что это не будет иметь места при трансляции выражения $a\times b\times c+d\times e\times f+g\times h\times i$.

5. Сколько адресов ячеек для запоминания промежуточных значений будет использовано в командах, возникающих при трансляции выражения $a\times b\times c+d\times e\times f+g\times h\times i$? Продумать способ экономии ячеек для запоминания промежуточных значений; эта экономия должна привести к тому, что для данного выражения потребуется только одна ячейка.

§ 4. Трансляция операторов и программы

Предложенный в § 2 алгоритм вычисления значения выражения удалось в § 3 превратить в алгоритм получения машинных команд вычисления этого значения. В настоящем параграфе область определения последнего алгоритма будет расширена: станет возможным применение алгоритма не только к выражению, но и к оператору, и к программе в целом. Как и полагается при трансляции, результатом применения алгоритма будет по-прежнему последовательность машинных команд, эквивалентная заданной алгольной конструкции*).

Оператор присваивания не доставляет трудностей. Присваивание переменной значения выражения выполняется после вычисления этого значения, и знаку := дается более низкий, в сравнении со знаками арифметических операций, приоритет. При выходе из M_1 знака := из M_2 извлекаются две очередные порции. Если первая порция состоит из одного слова и, значит, будет лишь заготовкой команды, то ее надо превратить в настоящую команду — записать в старшие разряды код операции считывания в сумматор. Вторая порция

*) Правомерна следующая точка зрения. Выполнение предложенного в § 3 алгоритма можно толковать в определенном смысле как вычисление значения выражения: значением выражения считается последовательность эквивалентных ему команд. Операторы алгольной программы в этом смысле тоже можно считать выражениями — им тоже соответствуют последовательности команд Таким образом, задача трансляции может рассматриваться как задача вычисления значения. Естественно пытаться вычислять это значение — последовательность команд — действиями с магазинами M_1 и M_2.

заведомо должна быть заготовкой, так как в левой части оператора присваивания всегда располагается переменная; в старшие разряды этой заготовки записывается код операции считывания с сумматора. Две порции после указанной доработки склеиваются.

В предположении, что для x отведена ячейка 01000, а для a, b, c — ячейки 01001, 01002 и 01003, оператор присваивания $x:=(a+b)\times c$ будет транслироваться в следующем порядке. В M_1 попадает заготовка 00001000, в M_2 — знак $:=$, они остаются в магазинах до окончания трансляции выражения $(a+b)\times c$, которая идет так, как говорилось в предыдущем параграфе. Затем заготовка 00001000 превращается в 01101000 и помещается вслед за последовательностью команд вычисления значения выражения. На верхушке M_1 появится порция

00000100
01201001
00101002
00301003
01101000

Как обычно, первое слово порции — это заголовок; предполагается, что при трансляции выражения $(a+b)\times c$ использовано решение задачи 3 предыдущего параграфа.

Трансляция составного оператора тоже очень проста: **begin** и **end** ведут себя как левая и правая скобки. Символ ; можно рассматривать как знак операции простого склеивания последовательностей команд, такая операция должна иметь приоритет более низкий, чем другие операции.

Рассмотрим теперь условный оператор, ограничась конструкцией

if B **then** C **else** D

Эту конструкцию надо рассматривать уже как трехместную операцию получения группы команд (до сих пор все операции были двухместными). Первая цель — разработка правил получения в M_1 групп команд, соответствующих B (см. задачу 5 § 2), C и D; знаком операции соединения этих групп будет **else** (**if** и **then** должны уничтожиться). Вот эти правила: символ **if**, попадая в M_1, ведет себя как левая скобка — ничего не выталкивает, **then** выталкивает все, вплоть до **if**.

Потом **if** уничтожается, а **then** попадает на верхушку. Символ **else** точно так же заменяет **then**. Попав в M_1, **else** получает приоритет более высокий, чем ; , и более низкий, чем := .

Все получающие приоритет знаки располагаются следующим образом по его убыванию:

$\uparrow$
$\times$, /
$+$, $-$
$<$, $>$, $\leqslant$, $\geqslant$, $=$, $\neq$
:=
else
;

(внутри каждой группы приоритеты одинаковы).

При соединении последовательностей команд для *B*, *C* и *D* (это соединение происходит при выходе символа **else** из M_1) нужна команда перехода, а в такой команде должен указываться адрес команды, к которой нужно перейти. Но этот адрес остается неизвестным, пока не оттранслирована вся программа. Указанная проблема возникает и при трансляции операторов перехода.

Обычный путь решения этой проблемы — помещение вместо команд перехода и тех команд, к которым надо переходить, некоторых заготовок. После получения с помощью магазинов M_1 и M_2 всей последовательности машинных слов, содержащей среди команд некоторое количество заготовок, эта последовательность просматривается и заготовки заменяются настоящими командами.

При построении заготовок и замене их командами составляются и используются некоторые таблицы.

Трансляция описания сводится к резервированию ячеек памяти. Каждому встреченному в программе числу тоже ставится в соответствие ячейка, в которую сразу же заносится изображающее это число машинное слово. Соответствие между переменными, числами и ячейками описывается таблицей.

Задачи.

1. Наряду с операторами алгола рассмотрим оператор обмена значениями между переменными, изображаемый $x \leftrightarrow y$, где x и y — переменные. Задать

приоритет знака ↔ в сравнении со знаками алгола и объяснить, как бы можно было транслировать оператор обмена значениями.

2. Разработать правила трансляции оператора цикла.

3. Кроме рассмотренной в тексте этого параграфа формы условного оператора, в алголе, как известно, допустима форма **if** *B* **then** *C*. Разработать общие правила трансляции условного оператора (**then** должен получить некоторый приоритет среди других знаков).

4. Почему после **then** правилами алгола запрещается помещать условный оператор и оператор цикла? (Какого рода неоднозначности могли бы возникнуть, если бы не было этого запрещения?)

5. Разработать намеченную в конце параграфа технику работы с заготовками команд перехода и тех команд, к которым надо переходить.

* *
*

После трансляции написанной на некотором алгоритмическом языке программы машина приступает к выполнению программы-результата. В следующей главе мы вернемся к понятию выполнения программы: это понятие будет расширено — станет возможным временное прерывание выполнения.

ГЛАВА V

ДИАЛОГОВЫЕ ПРОГРАММЫ. ОПЕРАЦИОННАЯ СИСТЕМА

Рассматриваются программы, с помощью которых человек может вести диалог с вычислительной машиной. Обсуждаются возможности использования машины во время ожидания реплик человека.

§ 1. Диалоговые программы

При выполнении программ возможен диалог (обмен сообщениями) человека с машиной. К диалоговым относятся, например, игровые программы, программы совместного решения задачи человеком и машиной и т. д. После вывода сообщения об очередном ходе или о результате этапа вычислений такие программы ждут от человека сообщения об ответном ходе или указаний относительно дальнейших действий. Программа ждет — значит, ее выполнение приостанавливается до получения сообщения человека.

Для написания диалоговых программ созданы специальные языки, предоставляющие много удобств, например обмен текстовыми сообщениями. Эти языки здесь рассматриваться не будут, так как речь пойдет только о принципе организации диалога человека и машины. Рассмотренный язык алгол становится пригодным для написания диалоговых программ, коль скоро принято следующее толкование оператора ввода. Если числа еще не подготовлены к моменту, когда надо выполнять этот оператор, то выполнение программы прерывается и продолжается только после появления нужных чисел. В следующих параграфах будет рассказано о выполнении таких программ машиной.

Разберем пример игровой программы. Игра в 100 спичек: из кучки, первоначально содержащей 100 спичек,

двое играющих поочередно берут от 1 до 10 спичек. Проигрывает взявший последнюю спичку. Для делающего ход невыгодными (т. е. проигрышными при безукоризненной игре противника) являются положения, когда в кучке содержится $11m+1$ спичек ($m=0, 1, 2, \ldots$). Это легко доказывается индукцией по m. При $m=0$ утверждение очевидно; если же $m>0$ и если взято t спичек ($1 \leqslant t \leqslant 10$), то противник может взять $11-t$ спичек: $1 \leqslant 11-t \leqslant 10$, и после этого приходится делать ход в положении, когда в кучке содержится $11(m-1)+1$ спичек.

Во всех других положениях делающий ход может легко выиграть — надо привести число спичек в кучке к виду $11m+1$.

Напишем реализующую эту стратегию программу. Человек и машина будут обмениваться числами, означающими число остающихся в кучке спичек. Первый ход будет за человеком. Машина всегда выигрывает, так как $100 = 11 \cdot 9 + 1$.

```
begin integer n, t, s;
    n := 100;
    repeat: input (t);
    if t = 0 then go to finish;
    s := 11 — (n — t); n := t — s;
    output (n); go to repeat;
    finish:
end
```

Задачи.

1. Сколько ходов будет сделано человеком в игре с машиной, выполняющей предложенную программу?

2. Переделать предложенную программу — включить проверку соблюдения человеком правил игры. Если правила нарушены, то должно выводиться то же число, что и перед неправильным ходом.

3. Составить программу, в которой начальное число спичек n задается человеком, а первый ход делается машиной. Может оказаться, что $n = 11m+1$ при некотором m, но вдруг человек в какой-то момент сделает не лучший ход.

4. Найти выигрышные и проигрышные положения в игре в n спичек при условии, что взявший последнюю спичку выигрывает.

5. Составить программу определения за 10 вопросов задуманного человеком целого числа от 1 до 1000. Каждый вопрос имеет вид: «Верно ли, что задуманное число больше числа k?», он задается выводом значения k. Ответом человека должно быть число 0 (нет) или 1 (да).

§ 2. Обмен сообщениями в диалоге

Упоминавшееся в гл. III устройство ввода с перфокарт непригодно для ввода сообщений человека при выполнении диалоговых программ: в момент обращения к этому устройству перфокарты должны быть на месте, и ожидание невозможно. Если бы даже ожидание и оказалось возможным, использование устройства ввода стало бы очень неудобным из-за длительности подготовки перфокарт.

При выполнении диалоговых программ связь человека с машиной происходит через *видеотерминал*, имеющий клавиатуру пишущей машинки и телевизионный экран, на котором высвечиваются последовательности знаков. Видеотерминал служит и устройством ввода, и устройством вывода — сообщения набираются человеком на клавиатуре, ответы читаются на экране.

Пока человек набирает свое сообщение, программа не выполняется. Ожидание заканчивается только после нажатия специальной клавиши, которая есть на клавиатуре видеотерминала, — клавиши исполнения.

Обсудим средства взаимодействия машины с несколькими видеотерминалами. Каждый видеотерминал связан с машиной через отдельный *канал ввода-вывода*. Каналы пронумерованы числами 1, 2, 3, ... После нажатия клавиши исполнения в машину поступает сигнал, что через данный канал можно прочесть сообщение. Входной язык машины должен содержать операцию проверки готовности канала. В нашей модели машины эта операция имеет код 061; в адресной части команды указывается номер (в восьмеричной системе) канала ввода-вывода. В результате выполнения этой операции в сумматоре появляется нулевое слово, если сообщение не готово, и появляется слово 77777777 в противном случае.

Для ввода и вывода можно использовать прежние операции с кодами 031 и 032. Дополнительно определим операцию с кодом 062 заявки на канал. После

выполнения, например, команды 06200007 каждая операция ввода и вывода будет использовать седьмой канал. Так будет продолжаться до следующего выполнения команды с кодом операции 062.

После ввода с видеотерминала в сумматоре оказывается (записанное в виде заголовка) число введенных машинных слов. Нулевая адресная часть после кода операции заявки на канал означает заявку на ввод с перфокарт и вывод на бумагу с помощью тех устройств, о которых говорилось в гл. III. Эти же устройства подразумеваются в операциях ввода и вывода до первого появления в программе команды с кодом операции 062.

Выполнение следующих команд приводит к ожиданию появления сообщения в канале ввода-вывода с номером 1, чтению этого сообщения и выводу его через канал с номером 5.

```
00100 | 06100001 |
00101 | 02300100 | } ожидание готовности 1-го канала
00102 | 06200001 |   заявка на 1-й канал
00103 | 03100177 |   ввод сообщения
00104 | 06200005 |   заявка на 5-й канал
00105 | 03200177 |   вывод сообщения
```

При решении задач этого параграфа предлагается придерживаться алголоподобной формы записи программ, используя, кроме обычных операторов и функций алгола, оператор заявки на канал *dc*(*n*) и функцию проверки готовности канала *ready*(*n*) (*n* — это номер канала). Функция *ready*(*n*) принимает значение 1, если через канал с номером *n* уже можно прочесть сообщение, и 0 в противном случае. Перепишем приведенную последовательность команд (передаваемое сообщение считаем целым числом).

```
begin integer k;
   L: if ready (1) = 0 then go to L;
      dc (1); input (k);
      dc (5); output (k)
end
```

Задачи.

1. Справочная служба дает ответы на поступающие из разных мест вопросы. Упрощенная модель: вопрос представляется целым числом от 1 до *n*, ответ — тоже целое число (обозначим ответ на вопрос *i* через a_i). Со-

ставить программу ответа на вопросы, поступающие через каналы с номерами 1, 2, ..., m. При каждом получении вопроса i через тот же канал должен выдаваться ответ a_i. Каналы просматриваются в следующем порядке: 1, 2, ..., m, 1, 2, ..., m, 1, 2, ..., m, ... Если вопрос через некоторый канал еще не задан, то этот канал пропускается. Просмотр каналов заканчивается, как только через какой-нибудь канал поступит не лежащее в диапазоне от 1 до n число.

2. Условие предыдущей задачи изменяется: прервать опрос (просмотр) каналов можно только через 1-й канал. Если число, не лежащее в диапазоне от 1 до n, поступает через какой-нибудь другой канал, то это число возвращается через тот же канал вместо ответа.

3. Упрощенная модель системы продажи авиационных и железнодорожных билетов похожа на упрощенную модель справочной службы (см. предыдущие задачи). Заказ — это два числа i, j: первое отражает дату, номер рейса и тип места, второе — количество билетов. Если $j \leqslant a_i$, то должно быть выведено число 1 (заказ выполнен), а значение a_i уменьшено на j. В противном случае должен быть выведен 0. Остановка происходит после получения через первый канал чисел 0, 0.

4. В модели системы продажи билетов закрепить за первым каналом функцию возврата билетов. Таким образом, получение через первый канал чисел i, j ведет к увеличению a_i на j.

5. В модель системы продажи билетов внести еще одно изменение: канал возврата билетов должен опрашиваться существенно чаще других. Рассмотреть две возможности: а) порядок опроса всех каналов берется таким: 1, 2, 1, 3, ..., 1, m, 1, 2, 1, 3, ... ; б) порядок опроса: 1, 2, ..., m, 1, 2, ..., m, ..., но перед тем, как дать отрицательный ответ на некоторый запрос, обращаются на всякий случай дополнительно к каналу возврата.

§ 3. Многопрограммный режим работы машины

Во время ожидания сообщений при выполнении диалоговой программы машина работает вхолостую. Хорошо бы в этом случае заставить машину выполнять другие программы. Это простое соображение может быть реализовано, только если машина оснащена

специальной аппаратурой и если входной язык машины содержит некоторые специальные операции, которые позволяют составить *операционную систему* — управляющую программу, обеспечивающую *многопрограммный режим* работы машины.

Опишем средства, необходимые для создания операционной системы.

Внешняя память. Вычислительная машина снабжается дополнительными хранилищами данных — *дисками.* Каждый диск может вместить в несколько раз больше слов, чем память машины. Внешняя память состоит из нескольких дисков. Из памяти машины с помощью специальных операций содержимое ряда последовательных ячеек может быть переписано на диск. Аналогично, порция машинных слов может быть считана с диска в память машины. Эти операции называются операциями обмена с внешней памятью; они используются не только в операционной системе, но и в программах обработки больших массивов, которые могут вызываться в память и обрабатываться по частям.

Обмен с внешней памятью идет гораздо быстрее, чем ввод с перфокарт, вывод на бумагу, чтение с экрана и вывод на экран видеотерминала. Правда, обмен совершается довольно большими порциями: например, по 1024 машинных слова.

Прерывания. Примем, что наша модель машины, как и многие настоящие машины, будет находиться в одном из двух состояний и в зависимости от состояния по-разному выполнять программы.

В первом состоянии программы выполняются так, как было показано в гл. III. В этом состоянии машина выполняет команды операционной системы. Будем предполагать, что операционная система постоянно занимает ячейки 70000—77777.

Во втором состоянии машина, в частности, особым образом реагирует на команды ввода и остановки: вместо ввода или остановки происходит изменение состояния и безусловный переход к команде в некоторой оговоренной ячейке, например в ячейке 70000. Продолжительность каждого пребывания во втором состоянии не превышает некоторого фиксированного времени. Мы будем считать, что это время — секунда. Если по прошествии секунды выполнение программы не завершилось и не встретились команды ввода, то все равно выполнение

программы прерывается, изменяется состояние машины и происходит безусловный переход к команде в ячейке 70000. Во время прерывания в ячейку 77777 попадает адрес команды, на который прервалось выполнение программы. Переход машины из первого состояния во второе может произойти только в результате выполнения специальной команды.

Все программы, кроме операционной системы, выполняются машиной во втором состоянии; будем пока считать, что они не затрагивают ячеек 70000—77777, позднее от этого предположения мы откажемся.

Суть многопрограммного режима: машина выполняет каждую программу не сразу от начала до конца, а поэтапно. Этап выполнения программы продолжается до прерывания, после которого наступает черед операционной системы: она переписывает во внешнюю память все, что имело отношение к выполнявшейся программе, и восстанавливает (переписью из внешней памяти) содержимое ячеек, сумматора и регистров таким, каким оно было после прерывания выполнения другой программы. Из первого состояния машина переключается во второе, и загруженная программа начинает выполняться с нужной команды.

Программы загружаются операционной системой в некоторой очередности. Если диалоговая программа ждет сообщения человека, то до готовности сообщения ее очередь пропускается. Вместо уже выполненных программ берутся новые программы, которые вводятся во внешнюю память заранее в достаточно большом количестве.

В многопрограммном режиме машина вперемешку выполняет и диалоговые, и обычные программы. Прерывания по времени не позволят слишком долго задерживаться программам, предписывающим длительный счет без вводов величин, и диалоговым программам, самостоятельно проверяющим готовность сообщения в каналах и предусматривающим ожидание.

При выполнении машиной отдельной программы процессор не используется не только во время ожидания реплик человека, но и во время ввода данных с перфокарт и вывода на бумагу. И в этом случае операционная система не позволяет процессору простаивать и поручает ему выполнение других программ.

Операционная система устанавливает определенную иерархию программ; в частности, возобновление выполнения диалоговых программ происходит чаще, чем введенных с перфокарт. Благодаря частым проверкам готовности канала у работающего за видеотерминалом создается впечатление, что он работает на машине один.

Когда встречается программа, написанная на каком-нибудь алгоритмическом языке, на долю операционной системы выпадает извлечение сведений из внешней памяти и запуск нужного транслятора.

Не только трансляторы, но и сама операционная система не хранится постоянно целиком в памяти машины. Например, после окончания выполнения машинной программы операционная система обычно заставляет машину печатать некоторые сведения о проделанной работе — затраченное время, количество потребовавшейся бумаги и т. д. Часть операционной системы, которая подготавливает печать этих сведений, вызывается в память машины, только когда в этом есть надобность. Постоянно присутствующая в памяти машины часть операционной системы — ее иногда называют резидентом системы — составляет небольшую долю всей системы. Вся операционная система — программа, полностью регламентирующая работу машины, — настолько велика, что не может целиком уместиться в памяти.

Хотя операционная система занимает часть памяти, это не мешает выполнению других программ благодаря мерам, принимаемым самой операционной системой. Часть выполняемой в данный момент программы и содержимое части ячеек, используемых этой программой, хранятся во внешней памяти. Когда надо, операционная система вызывает порции машинных слов из внешней памяти в память машины с предварительным освобождением места путем переписи во внешнюю память. Здесь есть весьма трудный момент пересчета адресов при выполнении программы.

Вообще, память машины, как физическое устройство, может содержать гораздо меньше ячеек, чем предполагается во входном языке машины. Операционная система расширяет физическую память машины за счет внешней памяти, но программист может этим не интересоваться (и может даже не знать о существовании внешней памяти), он имеет дело с так называемой прямоадресуемой памятью.

Задачи.

1. Каждый диск разбит на 1024 листа, и каждый лист может вместить 1024 машинных слова. Считывание в память машины и запись во внешнюю память происходят целыми листами. Будем предполагать, что в алгольных программах можно пользоваться операторами обмена *read* (n, m, k, a) и *write* (n, m, k, a); n — номер диска, m — номер начального листа, k — количество листов (считываемых или записываемых), a — идентификатор массива с количеством элементов, равным $1024k$.

Написать программу выбора наибольшего элемента из целых чисел, целиком заполняющих диск с номером 5. Числа вызывать в память машины порциями по 4 листа.

2. Используя операторы *read* и *write* (см. предыдущую задачу), написать программу выполнения следующего задания. Пусть диск с номером 5 заполнен неотрицательными целыми числами. Исключить из этих чисел все нули, оставив прежним порядок остальных чисел. Если среди данных чисел есть хотя бы один нуль, то в результате выполнения программы количество чисел уменьшится. Поместить в этом случае в конце получающегося набора чисел —1. Новый набор чисел должен занять тот же диск с номером 5. В память машины по-прежнему можно вызывать не более 4 листов.

3. Пусть машина выполняет в многопрограммном режиме несколько программ, одна из которых диалоговая, а выполнение остальных n программ не требует долгое время ввода и вывода величин. Пусть прерывание программы, загрузка и запуск новой программы занимают a секунд. Пусть для получения ответа на некоторое сообщение при выполнении диалоговой программы требуется, чтобы процессор проработал b секунд. Но из-за того, что машиной рассматриваются еще n упомянутых программ, время ожидания ответа равно c секундам. Определить время ожидания ответа на сообщение, требующее $b/2$ секунд работы процессора.

4 Выражаются ли однозначно величины b и c из предыдущей задачи друг через друга? Если да, то каким образом?

5. Пусть 071 — код операции переключения машины из первого состояния во второе, адресная часть соответствующей команды — нулевая. Написать последова-

тельность команд, выполнение которой приводит к переключению машины во второе состояние и переходу к команде, адрес которой получается увеличением на 1 адреса, записанного в адресной части ячейки 77777. Использовать регистры нельзя, так как их содержимое считается восстановленным таким, каким оно было до прерывания программы. Требуемая последовательность команд должна располагаться, начиная с ячейки 77000.

* *
*

Программы, предоставляющие различные удобства программисту и позволяющие рациональнее использовать вычислительную машину, образуют программное обеспечение машины. Кроме трансляторов и операционной системы к программному обеспечению относятся еще архивные системы («базы данных»), облегчающие работу с большими массивами данных во внешней памяти. К программному обеспечению относятся и диалоговые программы, позволяющие набором на видеотерминале некоторых инструкций извлекать из архивов во внешней памяти программы, редактировать их (исправлять ошибки или что-либо изменять), заставлять машину выполнять программы и т. д. Результат выполнения каждой инструкции появляется на экране видеотерминала.

ДОПОЛНЕНИЕ

О ДОКАЗАТЕЛЬСТВЕ СВОЙСТВ ПРОГРАММ

Во введении отмечалось, что программирование включает в себя и составление, и исследование программ. Мы обсудим сейчас один метод доказательства свойств программ. Каждое конкретное свойство, о котором будет идти речь, состоит в том, что если относительно исходных данных программы справедливо некоторое фиксированное утверждение, то относительно значений переменных в момент окончания выполнения программы справедливо некоторое другое фиксированное утверждение. Свойство программы может состоять, например, в том, что если исходное целочисленное значение n положительно, т. е. $n \geqslant 1$, то в момент окончания выполнения программы для значений переменных p, n выполнено $p = n!$. Если переменная n не изменяет своего значения при выполнении программы, то это свойство означает, что когда выполнение программы завершено, мы получаем значение $n!$; этим значением обладает переменная p. Это свойство принято записывать так: $\{n \geqslant 1\}\ P\{p = n!\}$, где под P понимается рассматриваемая программа. Обладает или нет программа подобным свойством — это может быть важным вопросом о правильности программы.

Ради простоты мы будем рассматривать только программы, описанные с помощью тех средств, которые были предложены в гл. I; более того, мы и из этих средств оставим не всё, ограничась инструкцией присваивания, условной и циклической инструкциями:

$x := E$, *пока* α *повторять* $(R_1; R_2; \ldots; R_k)$,
если β, *то* $(P_1; P_2; \ldots; P_m)$ *иначе* $(Q_1, Q_2, \ldots, Q_n)$,

где E — некоторое выражение, α и β — некоторые утверждения о значениях переменных (всюду далее такие утверждения обозначаются греческими буквами),

$R_1, R_2, \ldots, R_k, P_1, P_2, \ldots, P_m, Q_1, Q_2, \ldots, Q_n$ — инструкции этих же трех типов. Будет без оговорок предполагаться, что переменные принимают целочисленные значения. Записывая выражения, мы будем позволять себе употреблять традиционные математические обозначения операций (что не всегда возможно в алголе).

Уточним понятие свойства программы.

Определение 1. Программа P обладает свойством $\{\varphi\}P\{\psi\}$, где φ и ψ — некоторые утверждения о значениях, используемых в программе переменных, если каждым начальным значениям переменных, относительно которых справедливо φ, отвечают в момент окончания выполнения программы такие значения переменных, относительно которых справедливо ψ.

Простейшие примеры свойств программ:

$\{n = 0\}\, n := n + 1\, \{n = 1\}$,

$\{n \geqslant k,\ n^2 < 2k^2\}\, n := n + 1\, \{n \geqslant k + 1,\ (n-1)^2 < 2k^2\}$,

$\{n < m\}\, n := n + k\, \{n < m + k\}$,

$\{n < m + k\}\, n := 3n\, \{n < 3(m + k)\}$.

Менее простой пример дает упоминавшееся свойство программы вычисления факториала

$$\{n \geqslant 1\}\, p := 1;\quad m := 1;\ \textit{пока}\ m \neq n\ \textit{повторять}$$
$$(m := m + 1;\ p := p \cdot m)\, \{p = n!\}. \qquad (1)$$

Здесь мы видим не очень сложную программу, но тем не менее доказательство этого свойства уже доставляет хлопоты. С более сложной программой трудностей может быть больше. Как проводить подобные доказательства? Пересказ словами того, что происходит со значениями переменных при выполнении программы, может оказаться невразумительным. Идея метода, о котором пойдет речь — метода промежуточных утверждений, — состоит в сведении исходного свойства всей программы к некоторым свойствам составляющих ее инструкций. В конечном счете рассматриваемое свойство надо свести к группе свойств инструкций присваивания. И прежде всего, для этого оказывается полезной следующая

Теорема 1. *Пусть P и Q — программы, обладающие соответственно свойствами $\{\alpha\}P\{\beta\}$ и $\{\beta\}Q\{\gamma\}$. Тогда имеет место свойство $\{\alpha\}P; Q\{\gamma\}$ программы $P; Q$.*

Доказательство. Если относительно начальных значений переменных справедливо утверждение α, то по свойству $\{\alpha\}P\{\beta\}$ после выполнения P относительно значений этих переменных справедливо утверждение β, но по свойству $\{\beta\}Q\{\gamma\}$, если после P еще выполнить Q, то будет справедливо утверждение γ.

Пример. Программа $n := n + k$; $n := 3n$ обладает свойством

$$\{n < m\}\, n := n + k; \quad n := 3n\, \{n < 3(m + k)\}.$$

В самом деле, как уже отмечалось, имеют место свойства

$$\{n < m\}\, n := n + k\, \{n < m + k\}, \tag{2}$$

$$\{n < m + k\}\, n := 3n\, \{n < 3(m + k)\}. \tag{3}$$

Осталось применить теорему 1.

После рассмотрения этого примера становится понятным смысл названия «метод промежуточных утверждений». Между отдельными инструкциями мы фиксируем некоторые «промежуточные» утверждения, каждое из которых, как мы предполагаем, справедливо после выполнения одной инструкции и перед выполнением другой. Рассматривая последний пример, мы заметили, что после выполнения инструкции $n := n + k$ справедливо $n < m + k$. Удачный выбор промежуточных утверждений позволяет сводить свойство сложной программы к свойствам более простых. Скорее всего простейшей программой следует признать ту, которая состоит из одной инструкции присваивания. Свойства таких программ удобно доказывать, опираясь на следующую теорему.

Теорема 2. *Пусть после выполнения инструкции присваивания* $a := f(a, b, c, \ldots, x)$ *справедливо утверждение* $\beta(a, b, c, \ldots, x)$. *Тогда до выполнения этой инструкции было справедливо утверждение* $\alpha(a, b, c, \ldots, x) = \beta(f(a, b, c, \ldots, x), b, c, \ldots, x)$. *И наоборот — для любых значений* $a, b, c, \ldots, x$, *для которых справедливо* $\alpha(a, b, c, \ldots, x)$, *выполнение указанной инструкции присваивания приводит к такому изменению переменной* a, *что оказывается справедливым* $\beta(a, b, c, \ldots, x)$.

Доказательство. Пусть до выполнения инструкции присваивания $a := f(a, b, c, \ldots, x)$ переменные $a, b, c, \ldots, x$ имели значения $A_0, B_0, C_0, \ldots, X_0$, а после выполнения — $A_1, B_1, C_1, \ldots, X_1$. Ясно, что $A_1 = f(A_0, B_0, C_0, \ldots, X_0)$, $B_1 = B_0$, $C_1 = C_0$, $\ldots$, $X_1 = X_0$.

Следовательно, если выполнено $\beta(A_1, B_1, C_1, \ldots, X_1)$, то выполнено $\beta(f(A_0, B_0, C_0, \ldots, X_0), B_0, C_0, \ldots, X_0)$. Наоборот, если до выполнения инструкции присваивания имело место $\beta(f(A_0, B_0, C_0, \ldots, X_0), B_0, C_0, \ldots, X_0)$, то после выполнения будет иметь место $\beta(A_1, B_0, C_0, \ldots, X_0) = \beta(A_1, B_1, C_1, \ldots, X_1)$.

Пример. Применим утверждение теоремы для доказательства уже упоминавшихся свойств простейших программ (2), (3). Подстановка в утверждение $n<3(m+k)$ вместо переменной n выражения $3n$ дает $3n < 3(m+k)$, т. е. $n < m + k$, что доказывает свойство (3). Аналогично доказывается (2).

Еще один пример. Имеет место следующее свойство:

$$\{c = a,\ d = b\}\ a := a + b;\ b := a - b;$$
$$a := a - b\ \{c = b,\ d = a\}, \qquad (4)$$

которое, в силу того, что переменные c и d не меняют значений при выполнении программы, означает, что выполнение выписанных трех инструкций присваивания приводит к обмену значениями между переменными a, b.

Докажем это свойство. По теореме 2 имеет место свойство

$$\{c = b,\ d = a - b\}\ a := a - b\ \{c = b,\ d = a\},$$

по этой же теореме имеет место свойство

$$\{c = a - b,\ d = b\}\ b := a - b\ \{c = b,\ d = a - b\}.$$

Следовательно, по теореме 1

$$\{c = a - b,\ d = b\}\ b := a - b;\ a := a - b\ \{c = b,\ d = a\}.$$

По теореме 2:

$$\{c = a,\ d = b\}\ a := a + b\ \{c = a - b,\ d = b\},$$

следовательно, по теореме 1 имеет место (4).

Иногда, для того чтобы доказать некоторое свойство, удобно перейти к более сильному свойству и доказывать его.

Теорема 3. *Пусть имеет место свойство* $\{\alpha\}P\{\beta\}$. *Пусть утверждения* γ и δ *таковы, что* α *является следствием* γ (последнее понимается в том смысле, что если для значений переменных справедливо утверждение γ, то для них заведомо справедливо и α; обозначаться это будет так: $\gamma \Rightarrow \alpha$), а δ — *следствием* β. *Тогда имеет место* $\{\gamma\}P\{\delta\}$.

Доказательство следует из определения свойства программы.

Пусть мы хотим доказать свойство

$$\{n < m\}\, n := n + m\, \{n \leqslant 2m\}; \qquad (5)$$

по теореме 2 имеет место $\{n \leqslant m\}\, n := n + m\, \{n \leqslant 2m\}$; так как $(n < m) \Rightarrow (n \leqslant m)$ и $(n \leqslant 2m) \Rightarrow (n \leqslant 2m)$, то по теореме 3 имеет место (5).

Другое доказательство: по теореме 2 имеет место $\{n < m\}\, n := n + m\, \{n < 2m\}$, так как $(n < m) \Rightarrow (n < m)$ и $(n < 2m) \Rightarrow (n \leqslant 2m)$, то по теореме 3 имеет место (5).

Перейдем к условным инструкциям. Надо как-то свести свойство

$$\{\varphi\}\ \textit{если}\ \beta,\ \textit{то}\ (P)\ \textit{иначе}\ (Q)\ \{\psi\} \qquad (6)$$

к свойствам программ P и Q.

Теорема 4. *Для того чтобы имело место свойство* (6), *необходимо и достаточно, чтобы имели место свойства* $\{\varphi, \beta\} P \{\psi\}$ *и* $\{\varphi, \overline{\beta}\} Q \{\psi\}$.

Примечание. Утверждение $\overline{\beta}$ (читается «не β») состоит в том, что β не справедливо. Например, $\overline{a > b}$ эквивалентно $a \leqslant b$.

Доказательство теоремы. Необходимость: пусть имеет место свойство (6); если справедливо β, то, по определению условной инструкции, выполняется инструкция P и, следовательно, должно быть выполнено $\{\varphi, \beta\} P \{\psi\}$. Аналогично рассматривается случай, когда β не справедливо, т. е. справедливо $\overline{\beta}$. Достаточность: пусть значения переменных таковы, что справедливы φ и β; при выполнении условной инструкции *если* β *то* (P) *иначе* (Q) дело сведется к выполнению P и, поскольку имеет место свойство $\{\varphi, \beta\} P \{\psi\}$, то после выполнения условной инструкции значения переменных будут удовлетворять ψ. Аналогично рассматривается случай, когда первоначально справедливы φ и $\overline{\beta}$.

В качестве примера применения теоремы 4 докажем свойство

$$\{x, y - \textit{любые}\}\ \textit{если}\ x > y,\ \textit{то}\ (m := x)$$
$$\textit{иначе}\ (m := y)\ \{m \geqslant x,\ m \geqslant y\} \qquad (7)$$

для этого рассмотрим в соответствии с теоремой 4 свойства

$$\{x, y - \textit{любые},\ x > y\}\, m := x\, \{m \geqslant x,\ m \geqslant y\},$$

$$\{x, y - \textit{любые},\ \overline{x > y}\}\, m := y\, \{m \geqslant x,\ m \geqslant y\},$$

которые можно переписать так:

$$\{x > y\}\, m := x\, \{m \geqslant x,\ m \geqslant y\}, \tag{8}$$

$$\{x \leqslant y\}\, m := y\, \{m \geqslant x,\ m \geqslant y\}. \tag{9}$$

Применим к первому из этих двух свойств теорему 2. Если подставить в $m \geqslant x$, $m \geqslant y$ переменную x вместо переменной m, то получим $x \geqslant x$, $x \geqslant y$, что эквивалентно $x \geqslant y$. Поскольку $x > y \Rightarrow x \geqslant y$, то по теореме 3 действительно имеет место свойство (8). Таким же образом (даже не прибегая к теореме 3) можно доказать (9).

Циклическая инструкция — это наиболее интересный и трудный случай. Рассмотрим подробно свойство (1). Очевидно, что имеет место свойство $\{n \geqslant 1\}\ p := 1;\ m := 1\, \{n \geqslant 1,\ p = 1,\ m = 1\}$, поэтому, по теореме 1, для доказательства (1) достаточно было бы доказать свойство

$$\{n \geqslant 1,\ p = 1,\ m = 1\}\ \textit{пока}\ m \neq n\ \textit{повторять}$$
$$(m := m + 1;\ p := p \cdot m)\, \{p = n!\}. \tag{10}$$

Для этого естественно попытаться провести индукцию по числу этапов выполнения циклической инструкции. Если число этапов равно нулю, то все очень просто: раз одновременно выполнено $n \geqslant 1$, $p = 1$, $m = 1$, $m = n$, то $m = n = p = 1$ и, конечно, $p = n!$. Однако с переходом от числа этапов, равного $t\,(t \geqslant 0)$, к числу этапов, равному $t + 1$, дело обстоит хуже: если мы вынесем один этап за пределы цикла, т. е. заменим циклическую инструкцию последовательностью инструкций

$$m := m + 1;\ p := p \cdot m;\ \textit{пока}\ m \neq n\ \textit{повторять}$$
$$(m := m + 1;\ p := p \cdot m),$$

то не сможем воспользоваться предположением индукции, поэтому после выполнения присваиваний $m := m + 1$ и $p := p \cdot m$ уже не будет справедливым $m = 1$.

Для доказательства (10) имеет смысл воспользоваться теоремой 3 и перейти к некоторому другому свойству той же программы, заменив утверждение $n \geqslant 1$, $p = 1$, $m = 1$ некоторым его следствием α, таким, что имеет место

$$\{\alpha\}\, m := m + 1;\ p := p \cdot m\, \{\alpha\}.$$

Кроме этого, должно остаться в силе основание индукции: все должно быть верно для случая, когда число этапов равно нулю, т. е. для случая $m = n$.

Нетрудно убедиться, что в качестве α можно взять $p = m!$. Если $m = n$, то из $p = m!$ следует $p = n!$ — это основание индукции. Далее, очевидно, имеет место свойство

$$\{p = m!\}\, m := m + 1;\ p := p \cdot m\, \{p = m!\}.$$

Теперь ясно, что верна следующая теорема о циклической инструкции.

Теорема 5. *Пусть для утверждения* δ *выполнено*

1) $\alpha \Rightarrow \delta$,
2) $(\delta, \bar{\gamma}) \Rightarrow \beta$,
3) $\{\delta\}\, P\, \{\delta\}$.

Тогда имеет место свойство

$$\{\alpha\}\ \textit{пока}\ \gamma\ \textit{повторять}\ (P)\ \{\beta\} \tag{11}$$

циклической инструкции

$$\textit{пока}\ \gamma\ \textit{повторять}\ (P). \tag{12}$$

Утверждение δ, упоминаемое в теореме, оказывается справедливым непосредственно перед каждым выполнением и непосредственно после каждого выполнения P; в этом смысле δ — промежуточное утверждение.

Теорему 5 можно усилить, заменив условие 3 условием

$$3')\quad \{\delta, \gamma\}\, P\, \{\delta\},$$

так как, выполняется только когда справедливо γ.

Определение 2. Утверждение δ, удовлетворяющее условию $\{\delta, \gamma\}\, P\, \{\delta\}$, называется инвариантом циклической инструкции (12).

Таким образом, для доказательства свойства (11) достаточно подобрать удовлетворяющий условиям 1, 2 инвариант циклической инструкции (12). Свойство 3 или 3′ можно бы назвать законом сохранения справедливости утверждения γ.

Два примера свойств циклических инструкций и инвариантов, подходящих для доказательства этих свойств:

$$\{n \geqslant 1,\ p = 1,\ m = n\}\ \textit{пока}\ m \neq 0\ \textit{повторять}$$
$$(p := p \cdot m;\ m := m - 1)\, \{p = n!\}, \tag{13}$$

подходящий инвариант — $p \cdot m! = n!$;

$$\{k \geqslant 1,\ m = 1,\ n = k\}\ \textit{пока}\ n \neq 0\ \textit{повторять}$$
$$(m := 2m;\ n := n - 1)\, \{m = 2^k\}, \tag{14}$$

подходящий инвариант — $m2^n = 2^k$.

Ясно, что доказательство свойства (11) с привлечением инварианта δ есть, по существу, доказательство индукцией по числу шагов. Но интересно, что проверяя условия 1, 2, 3 или 1, 2, 3′, мы не рассматриваем явно и не упоминаем это число.

Примечание. Исследования описанного метода доказательства свойств циклических инструкций показывают, что в некоторых случаях нужный инвариант оказывается довольно сложным, и для его описания в свою очередь требуется циклическая инструкция, или нечто подобное. Проверка условий 1, 2, 3 или 1, 2, 3′ может стать задачей, эквивалентной по сложности исходной задаче. Но, с другой стороны, как показывают обсуждавшиеся примеры, применение этого метода часто выглядит вполне естественным и не вызывает затруднений.

Надо отметить, что мы столкнемся с существенными осложнениями, если попытаемся свести к свойствам отдельных инструкций свойство такой программы, в которой использованы переходы (вроде алгольных **go to**) — выполнение переходов не ведет к изменению значений переменных, но их, конечно, нельзя игнорировать при доказательстве свойств программы. Эти осложнения были одной из причин развития в программировании направления, именуемого структурным программированием, одним из принципов которого является отказ от использования переходов в программах. Доказано, что без переходов действительно можно обойтись, если в распоряжении имеются циклические и условные инструкции. С другой стороны, разработаны варианты метода промежуточных утверждений, позволяющие иметь дело с переходами.

Анализ циклической инструкции ставит еще одну задачу — распознавание завершимости выполнения этой инструкции. До сих пор мы считаем, что выполнение рассматриваемых программ заведомо завершается. Но в действительности выполнение циклических инструкций иногда не завершается, потому что для всех возникающих значений переменных справедливы утверждения, записанные после *пока*. Если завершимость выполнения программы не исследована, то доказанное свойство программы мало о чем говорит; вместо β в

$$\{n \geqslant 0\}\ \textit{пока}\ n \geqslant 0\ \textit{повторять}\ (n := n + 1)\{\beta\} \qquad (15)$$

можно подставить любое утверждение и получится некоторое свойство циклической инструкции. Это соответствует определению свойства программы, в котором есть слова «в момент окончания выполнения программы»; если бы мы захотели показать, что при некотором утверждении β свойство (15) не имеет места, мы должны были бы найти некоторое неотрицательное n, для которого выполнение циклической инструкции завершается, и после завершения не справедливо β, но такого n мы найти не можем.

Иногда для установления завершимости выполнения циклической инструкции достаточно рассмотреть характер изменения значений какой-нибудь переменной. Циклическая инструкция, входящая в (13), выполнится в n этапов — это видно из рассмотрения изменения значений переменной n. Циклическая инструкция, входящая в (14), выполнится в k этапов. Выполнение циклической инструкции, входящей в программу (1), завершится в n этапов. Но в более сложных программах значения переменных при выполнении могут изменяться немонотонно. Рассмотрим следующий пример:

пока $n > 1$ *повторять* (16)

(*если* n — *четное, то* $(n := n/2)$ *иначе* $(n := n + 1)$)

По ходу выполнения этой циклической инструкции значение переменной n может и уменьшаться, и увеличиваться. Оказывается, что выполнение этой циклической инструкции завершается при любом целом значении n.

Для доказательства завершимости выполнения в подобных случаях рассматривают значения не какой-то отдельной переменной, а значения некоторой функции, зависящей от переменных.

Теорема 6. *Пусть целочисленная функция* f, *зависящая от некоторых переменных программы* (12), *удовлетворяет следующим условиям*: 1) *ее значение положительно, если относительно значений переменных справедливо утверждение* γ; 2) *она убывает при изменении значений переменных в результате выполнения* P. *Тогда выполнение* (12) *завершается.*

Доказательство следует из того, что не существует бесконечной убывающей целочисленной последовательности с положительными элементами. Число этапов выполнения циклической инструкции не превзойдет значения f от исходных значений переменных.

Подбор функции f, как и подбор инварианта, требует изобретательности. Вернемся к программе (16). Функцию $f(n)$ определим для всех $n \geqslant 1$ следующим образом:

$$f(n) = \begin{cases} 0, & \text{если } n = 1, \\ n - (-1)^n, & \text{если } n > 1. \end{cases}$$

Проверим условия 1, 2, приведенные в теореме 6.

1) Если $n > 1$, то $n - (-1)^n \geqslant n - 1 > 0$.

2) Рассмотрим раздельно случаи, когда n четно и когда нечетно. Первый случай. Если $n > 2$, то $f(n/2) \leqslant \leqslant n/2 + 1 < f(n)$. Если $n = 2$ и $\frac{n}{2} = 1$, то $f(n/2) = = 0 < f(n)$. Второй случай: $f(n) = n + 1$, $f(n+1) = n$.

До сих пор не доказана и не опровергнута завершимость выполнения

пока $n > 1$ *повторять*

(*если* n — *четное, то* $(n := n/2)$ *иначе* $(n := 3n + 1)$)

для любого натурального n.

Задачи

1. Пользуясь теоремами 1, 2, доказать следующее свойство:

$\{c = a, d = b\}\, e := a;\ a := b;\ b := e\, \{c = b, d = a\}$.

2. Пользуясь теоремами 1, 2, выяснить, для каких исходных значений a и b после выполнения программы

$a := a - b;\ b := a - b$

переменные a и b будут иметь равные значения.

3. Вывести необходимые и достаточные условия того, что имеет место свойство

$\{\varphi\}$ *если* β, *то* $(P)\{\psi\}$.

4. Пользуясь только теоремой 5, доказать, что при любом φ имеет место следующее свойство:

$\{\varphi\}$ *пока* β *повторять* $(P)\{\overline{\beta}\}$.

5. Доказать, что выполнение циклической инструкции

пока $n > 1$ *повторять*

(*если* n — *четное, то* $(n := n/2 + 1)$ *иначе* $(n := n + 1)$)

не завершается ни для какого натурального $n > 1$.

ЛИТЕРАТУРА ДЛЯ ДАЛЬНЕЙШЕГО ЧТЕНИЯ

1. Лавров С. С. Универсальный язык программирования. — М.: Наука, 1972. (Полное изложение языка алгол 60.)
2. Пратт Т. Языки программирования: разработка и реализация. — М.: Мир, 1979. (Обзор наиболее распространенных современных алгоритмических языков.)
3. Нивергельт Ю., Фаррар Дж., Рейнгольд Э. Машинный подход к решению математических задач. — М.: Мир, 1977. (Алгоритмическая сторона различных математических задач.)
4. Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов. — М.: Мир, 1979. (Вопросы построения экономических алгоритмов; математические методы анализа алгоритмов.)
5. Журнал «Квант», начиная с № 9, 1979. (В № 9, 1979 г. открыт постоянный раздел «Искусство программирования».)
6. Ершов А. П. Введение в теоретическое программирование (беседы о методе). — М.: Наука, 1977. (Подробно рассматриваются две классические задачи теоретического программирования.)
7. Вирт Н. Систематическое программирование; введение. — М.: Мир, 1977. (Вводный курс программирования; значительное внимание уделяется доказательству правильности предлагаемых программ.)

Сергей Александрович Абрамов
ЭЛЕМЕНТЫ ПРОГРАММИРОВАНИЯ

(Серия: «Популярные лекции по математике»)

Редакторы *Н. Н. Васина, В. Д. Поддергина*
Технический редактор *Е. В. Морозова*
Корректоры *Т. С. Плетнева, И. Д. Дорохова*

ИБ № 11998

Сдано в набор 22.04.81. Подписано к печати 25.01.82. Т-03328. Формат 84×108¹/₃₂. Бумага тип. № 1. Литературная гарнитура. Высокая печать. Усл. печ. 5,04. Уч.-изд. 4.7. Тираж 200 000 экз. Заказ № 1178. Цена 15 коп.

Издательство «Наука»
Главная редакция
физико-математической литературы
117071, Москва, В-71, Ленинский проспект, 15

Ленинградская типография № 2 головное предприятие ордена Трудового Красного Знамени Ленинградского объединения «Техническая книга» им. Евгении Соколовой Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли. 198052, г. Ленинград, Л-52, Измайловский проспект, 29

22180/1
1020

1614/17

15 коп.

ПОПУЛЯРНЫЕ ЛЕКЦИИ ПО МАТЕМАТИКЕ

Вып. 1. **А. И. Маркушевич.** Возвратные последовательности.
Вып. 2. **И. П. Натансон.** Простейшие задачи на максимум и минимум.
Вып. 3. **И. С. Соминский.** Метод математической индукции.
Вып. 4. **А. И. Маркушевич.** Замечательные кривые.
Вып. 5. **П. П. Коровкин.** Неравенства.
Вып. 6. **Н. Н. Воробьев.** Числа Фибоначчи.
Вып. 7. **А. Г. Курош.** Алгебраические уравнения произвольных степеней.
Вып. 8. **А. О. Гельфонд.** Решение уравнений в целых числах.
Вып. 9. **А. И. Маркушевич.** Площади и логарифмы.
Вып. 10. **А. С. Смогоржевский.** Метод координат.
Вып. 11. **Я. С. Дубнов.** Ошибки в геометрических доказательствах.
Вып. 12. **И. П. Натансон.** Суммирование бесконечно малых величин.
Вып. 13. **А. И. Маркушевич.** Комплексные числа и конформные отображения.
Вып. 14. **А. И. Фетисов.** О доказательствах в геометрии.
Вып. 15. **И. Р. Шафаревич.** О решении уравнений высших степеней.
Вып. 16. **В. Г. Шерватов.** Гиперболические функции.
Вып. 17. **В. Г. Болтянский.** Что такое дифференцирование?
Вып. 18. **Г. М. Мнакьян.** Прямой круговой цилиндр.
Вып. 19. **Л. А. Люстерник.** Кратчайшие линии.
Вып. 20. **А. М. Лопшиц.** Вычисление площадей ориентированных фигур.
Вып. 21. **Л. И. Головина и И. М. Яглом.** Индукция в геометрии.
Вып. 22. **В. Г. Болтянский.** Равновеликие и равносоставленные фигуры.
Вып. 23. **А. С. Смогоржевский.** О геометрии Лобачевского.
Вып. 24. **Б. И. Аргунов и Л. А. Скорняков.** Конфигурационные теоремы.
Вып. 25. **А. С. Смогоржевский.** Линейка в геометрических построениях.
Вып. 26. **Б. А. Трахтенброт.** Алгоритмы и машинное решение задач.
Вып. 27. **В. А. Успенский.** Некоторые приложения механики к математике.
Вып. 28. **Н. А. Архангельский и Б. И. Зайцев.** Автоматические цифровые машины.
Вып. 29. **А. И. Костовский.** Геометрические построения одним циркулем.
Вып. 30. **Г. Е. Шилов.** Как строить графики.
Вып. 31. **А. Г. Дорфман.** Оптика конических сечений.
Вып. 32. **Е. С. Вентцель.** Элементы теории игр.
Вып. 33. **А. С. Барсов.** Что такое линейное программирование.
Вып. 34. **Б. Е. Маргулис.** Системы линейных уравнений.
Вып. 35. **Н. Я. Виленкин.** Метод последовательных приближений.
Вып. 36. **В. Г. Болтянский.** Огибающая.
Вып. 37. **Г. Е. Шилов.** Простая гамма (устройство музыкальной шкалы).
Вып. 38. **Ю. А. Шрейдер.** Что такое расстояние?
Вып. 39. **Н. Н. Воробьев.** Признаки делимости.
Вып. 40. **С. В. Фомин.** Системы счисления.
Вып. 41. **Б. Ю. Коган.** Приложение механики к геометрии.
Вып. 42. **Ю. И. Любич и Л. А. Шор.** Кинематический метод в геометрических задачах.
Вып. 43. **В. А. Успенский.** Треугольник Паскаля.
Вып. 44. **И. Я. Бакельман.** Инверсия.
Вып. 45. **И. М. Яглом.** Необыкновенная алгебра.
Вып. 46. **И. М. Соболь.** Метод Монте-Карло.
Вып. 47. **Л. А. Калужнин.** Основная теорема арифметики.
Вып. 48. **А. С. Солодовников.** Системы линейных неравенств.
Вып. 49. **Г. Е. Шилов.** Математический анализ в области рациональных функций.
Вып. 50. **В. Г. Болтянский, И. Ц. Гохберг.** Разбиение фигур на меньшие части.
Вып. 51. **Н. М. Бескин.** Изображения пространственных фигур.
Вып. 52. **Н. М. Бескин.** Деление отрезка в данном отношении.
Вып. 53. **Б. А. Розенфельд и Н. Д. Сергеева.** Стереографическая проекция.
Вып. 54. **В. А. Успенский.** Машина Поста.
Вып. 55. **Л. Беран.** Упорядоченные множества.
Вып. 56. **С. А. Абоамов.** Элементы программирования.
Вып. 57. **В. А. Успенский.** Теорема Гёделя о неполноте.